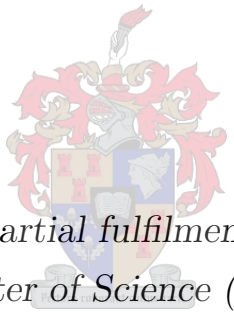
Explaining neural networks used for modeling credit risk.

by

Zhunaid Mohamed



Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Computer Science) in the Faculty of Science at Stellenbosch University

Supervisor: Prof. W. Visser

Co-supervisor: Prof. B. Herbst, Dr. M. Hoffman

March 2021

The financial assistance of the National Research Foundation(NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2021/03 .....

Abstract

Explaining neural networks used for modeling credit risk.

Z. Mohamed

Department of Computer Science,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MSc (CS)

March 2021

Calculating risk before providing loans is a common problem that credit companies face. The most common solution is credit employees manually assessing the risk of a client by reviewing their credit portfolios. This can be a slow process and is prone to human error. Recently credit companies have been adopting machine learning techniques in order to automate this process, however this has been limited to linear techniques due to interpretability being a strict requirement. Neural networks could provide significant improvements to the way credit risk is modeled, however these are still seen as black boxes. In this work we compare various techniques which claim to provide interpretability into these black boxes. We also use these techniques to provide explanations on a neural network trained on credit data that has been provided to us.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning techniques have become widely adopted in many different domains due to their potential to produce fast and accurate results. Some of these techniques provide on par or even surpass the predictive accuracy provided by humans. However it is often difficult to tell whether the techniques are correctly solving the problem or are exploiting artifacts in the data. In domains where interpretability is required for safety or legal reasons such as in medicine [1] where the model has to be proven trustworthy, there is often a trade off of accuracy for greater interpretability. Inherently interpretable models such as linear models where the weight coefficients can be considered an explanation or decision trees [2] which provide explicit rules are the techniques of choice. Neural Networks which were not considered in these domains due to them being considered a *black box* now have several different techniques which aim to provide interpretations into their decision making. This thesis highlights several techniques that have been used to explain the decision making of neural networks and 3 in particular will be discussed and evaluated in detail.

1.1 Problem Statement

Suppose you are approached by a real estate agency to build a model that can predict the market price of a property given its features such as *size*, *location*, and *number of bedrooms*. Now for example say your model gives an estimated worth

of R1.5 million rand for the property, it is expected that a list of reasons be given as to why this property was valued as such. If we used a simple linear regression model for this, it would be as simple as extracting the weights for each feature and listing which features contributed the most to this price. However, after some experimentation you realize that the linear regression model is not accurate enough and by using modern neural networks you are able to greatly increase your accuracy. You are now faced with a problem since there is no simple way to explain what features were important in a neural network and are forced to use the inferior model due to its interpretability.

1.2 What is a neural network

The goal of a Neural Network is to simulate the ability of the human brain to detect patterns within data in order to make some decision. For example given a picture of a cat with the aim at determining whether it is a cat or a dog. By distinguishing certain features such as their ears and mouth shape we as humans are able to come to the decision that it is indeed a cat.

Figure 1.1 is the generic structure of a Feedfoward Neural Network. Each individual circle is a *neuron* which produces some real-valued activation. Sets of neurons are separated into *layers* which each have a different representation of the data. The arrows which interconnect layers are called *weights*. The input layer is what the Neural Networks sees as input data such as the pixels of an image of a cat. The hidden layers are internal representations of data that is learnt by the network. The output layer is the result of the decision of the network such as whether it believes it is a cat or a dog. In order to make decisions our Neural Network first has to be trained.

This entails giving the Neural Network a set of samples with their corresponding *correct decisions*. In the case of differentiating between cats and dogs, we may have a set of images of both cats and dogs and their corresponding labels. The weights and hidden neurons are initialized either randomly or by some algorithm. The input data is then put through the network and the *produced outputs* are compared to the *actual outputs*. The *error* is calculated which is the difference between the actual and produced outputs by calculating the gradients with an

algorithm called *backpropagation*[3][4]. Using this calculated error the parameters are updated within each layer using *gradient descent* [5]. This process is repeated until it is decided that the error is small enough according to some metric and the result is a trained Neural Network.

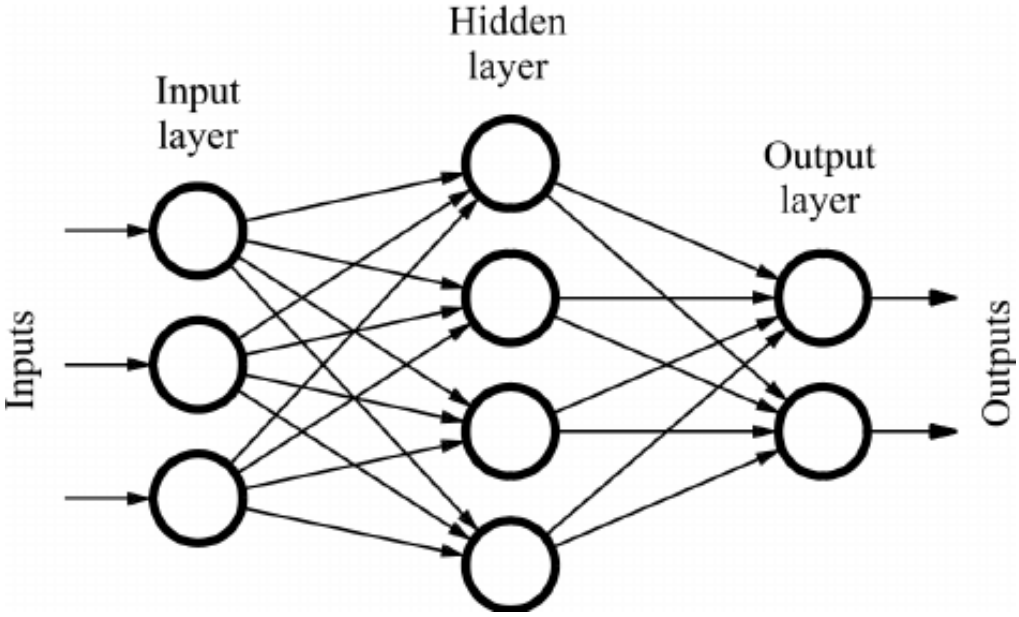


Figure 1.1: Architecture of a generic Feedforward Neural Network. **Image from** [6].

1.3 Why interpreting neural networks is difficult

Firstly we define exactly what we mean by *interpretability*. Interpretability is mapping a complex or abstract concept that is incomprehensible to humans into a domain that humans can understand [7]. Once we have an interpretable domain we want to produce an *explanation* which are the features which contributed towards a specific prediction [7]. Neural Networks are usually treated as *black boxes* of which we have little to no understanding of their inner workings.

An example is present in Figure 1.2 where an adversary [8] slightly adjusted the original image on the left by adding a form of noise which causes the Neural Network to misclassify the *Panda* as a *Gibbon*. From a human perspective both images are

obviously of a Panda but the Neural Network was easily fooled and there is no simple way to understand why. Looking at the architecture of a Neural Network

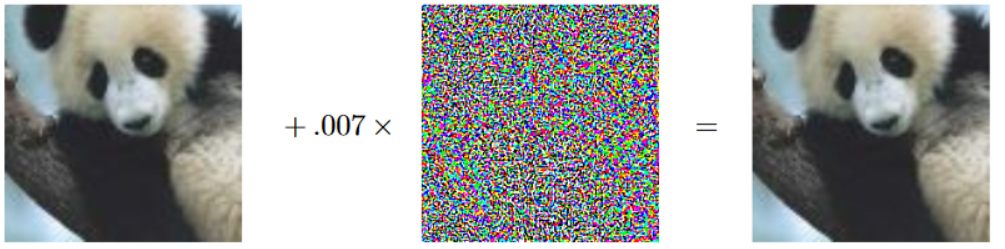


Figure 1.2: A demonstration showing that by adding a small amount of noise to the image of a panda we were able to fool the Neural Network into misclassifying it as a gibbon. **Image from** [9].

we could identify which individual neurons activate based on the presence of certain features, an example being the shape of the pandas nose in Figure 1.2. However, this does not seem to be true as both images contain the same features yet the Neural Network failed to correctly identify the panda. Therefore the idea of directly monitoring how neurons activate in response to features is not reliable. In the case of a linear model trained on images we could simply look at the relationship between the features extracted from algorithms such as SURF [10] and SIFT [11] and the output. This is not as simple when working with Neural Networks due to the hidden layers having their own transformations that are learnt through backpropagating the errors [3][4]. In the case of a *Convolutional Neural Network (CNN)* [12] which is an image-based network, simply viewing the hidden layers may look like noise. In a CNN the features which are extracted are learnt by the network itself and may result in the reliance on noise or artifacts within the image.

Another example can be seen in Figure 1.3, when changing the images to their negative the network is unable to discern the correct class. This shows that rather than specifics like edges and shapes the classifier could be relying on color specific artifacts in it's classification scheme. It is natural to think that by providing the

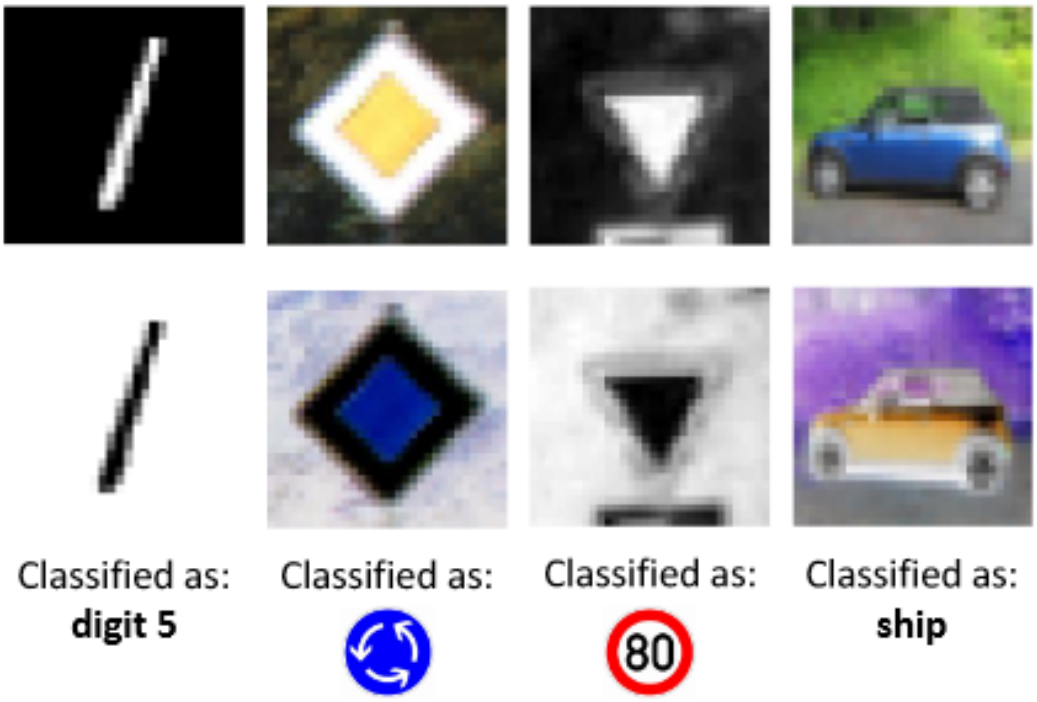


Figure 1.3: Simply making the images negative causes a misclassification by the network. **Image from** [13].

images altered by the noise into the training data we can prevent this form of misclassification. This is not feasible as we can not protect against every form of alteration. Therefore we need concrete explanations on how the input features affect the classification scheme of the network so that we may ascertain that the network has properly learnt actual features rather than artifacts.

This extends to other network architectures. For a *Recurrent Neural Network (RNN)* [12] trained on classyfing a movie review as either good or bad, by slightly altering the input sequences it was possible to have the network misclassify 100% of the training data [14]. For instance changing the review "I wouldn't rent this one even on dollar rental night." into "Excellent wouldn't rent this one even on dollar rental night." the network was fooled into misclassyfing the review as positive [14]. By simply inserting words with positive connotations into the input the RNN was mislead. The goal of providing a relationship of input features to prediction output is referred to as the *attribution problem* [15].

1.3.1 Overview of Interpretability

As we have discussed in the previous section, linear models can be easily explained since their prediction is simply a linear combination of feature values weighted by model coefficients. However due to the need for more powerful non-linear models *Random Forests* [16] among others became a popular choice. Random Forests is a technique that constructs multiple decision trees and outputs the prediction as the average prediction of the individual trees. Since this technique was non-linear it was not adopted into many fields due it being difficult to interpret. A PhD thesis by *Gilles Louppe* [17] provided a methodology for extracting the importance of features on a global level from Random Forests. The interpretability of Random Forests was further expanded when the popular machine learning library *scikit-learn* released a blog post on how to obtain importance of features for individual predictions [18].

With the sudden growth in popularity of Neural Networks, interpretation was once again needed before it could replace older and weaker techniques. There have been several techniques which aim to interpret Neural Networks which range from using decision trees to approximate the network to isolating individual neurons in an attempt to explain their importance. These techniques are discussed in more detail in Chapter 2. For this thesis we have chosen 2 tools which claim to be model-agnostic explainers which are explainers that can be used on any machine learning model, from linear models to Neural Networks and a tool which provides a deep framework to explore the inner workings of a Neural Network. These will be discussed in detail in Chapter 3.

1.4 Our approach

1.4.1 Discuss and Evaluate our chosen tools

We discuss and evaluate 3 chosen tools which aim to provide interpretation for Neural Networks decisions. We have selected popular machine learning problems which range from the popular *MNIST* digit classification problem to predicting whether a movie review posted on *IMDB* is positive or negative. We have trained models with varying architectures aimed at solving these problems. Using our 3

tools we attempt to provide explanations into these models in order to discern which features have the biggest impact in determining the prediction. We also compare how these explanations differ for different input types and whether they can be easily understood.

1.4.2 Generate explanations for a Neural Network used to solve a real world credit problem

The main contribution of this work is that we apply these techniques on a real world example. The credit space is a high risk environment and therefore any machine learning technique that is used has to be interpretable. Using a credit risk dataset provided to us by *Praelexis* we will train both a linear model, which is a commonly used solution, and a Neural Network. Using our model-agnostic tools, we will generate explanations for the Neural Network and compare them to the inherent explanations present in the linear network. Our goal is to determine whether the explanations for the Neural Networks are good enough to allow them to be used for credit risk modeling.

1.5 Thesis Goals

The goals we are aiming to achieve with this thesis are as follows:

- Evaluate the provided explanations between our 3 chosen tools on various machine-learning problems.
- Train a linear model and a Neural Network on the provided credit risk data.
- Provide explanations into the credit Neural Network and compare the interpretability to the linear model.
- Answer the question whether these explanations are sufficient enough to consider Neural Networks in domains where interpretability is a strict requirement.

1.6 Thesis Structure

Chapter 2 describes related techniques used for providing interpretability into Neural Networks.

Chapter 3 provides a detailed background into our 3 chosen tools by thoroughly explaining their inner workings.

Chapter 4 provides an evaluation and comparison between the explanations generated by our chosen tools on various popular machine learning problems.

Chapter 5 we train and evaluate a model trained on credit risk data and provide explanations using our tools.

Chapter 6 concludes our research.

Chapter 2

Related Work

Our 3 chosen tools includes 2 model-agnostic explainers, namely *LIME* [19] and *SHAP* [20]. As well as "a collection of infrastructure and tools for research in Neural Network interpretability" [21] known as *Lucid* . These tools will be discussed in greater detail in Chapter 3. In this chapter we will discuss related research efforts which attempt to provide interpretability into Neural Networks and explain why they have not been chosen for further evaluation.

2.1 Activation Maximization

Activation Maximization (AM) is the idea of generating input patterns that would maximize the activation of a given hidden unit in a Neural Network [22][23][24]. Suppose we had a Neural Network which maps a input vector x to a set of classes (w_c). The output layer of the Neural Network is a set of neurons which encode this as class probabilities $p(w_c|x)$. We can generate a prototype x^* of a class w_c by optimizing,

$$\max_x \log p(w_c|x) - \lambda||x||^2 \tag{2.1}$$

where the rightmost term is a regularizer which prefers inputs which are close to the origin. The probabilities produced by the Neural Network are functions which have gradients [4]. Therefore we are able to optimize (2.1) using gradient ascent [25]. This allows us to visualize the features that individual neurons are looking for within the input. For example given a network which attempts to discern

whether an image is of a cat or dog. By using AM we theoretically should be able to discern which neurons in the network look for features of a dog such as floppy ears. However image-based networks prototypes mostly look like gray images where key points have patterns [22]. The prototypes produced by this optimization are mostly unnatural and therefore can not be considered reliable. One of our chosen tools Lucid further expands on this concept for image-based networks where the prototypes produced are easily understood.

2.2 Sensitivity analysis

Sensitivity Analysis [7] is a technique used to identify the most important input features within a model [26]. We use the model's locally evaluated gradient to calculate the relevance scores of the features as,

$$R_i(x) = \left(\frac{\partial f}{\partial x_i}\right)^2 \tag{2.2}$$

where x is the input features, x_i is the feature at index i, and f is the model. From (2.2) we can see that the gradient is being evaluated at the data point x. The features which the output is most sensitive to are also considered the most relevant. Therefore a feature can be considered relevant if it is present in the data and has a large impact on the gradient. This tells us that a feature is relevant in some way to the model, but it does not tell us exactly how it affects the prediction.

2.3 Trepan

Extracting Tree-Structured Representations of Trained Networks (Trepan) [27] is an algorithm which attempts to extract comprehensible and symbolic representations from trained Neural Networks. This is done by inducing a decision tree [2] which is interpretable and describes the concept represented by the network. The goal of the algorithm is to produce a decision tree which given the same input as the Neural Network produces the same results. The decisions made by the decision tree can be seen as the same decisions that the Neural Network makes. Therefore

the interpretation of the decision tree can be seen as an approximate explanation of the network. This algorithm was developed in 1996 and there are not many working examples in practice. Popular tools have opted for using linear models as approximators due to them being easier to compute. Due to modern Neural Networks having become more complex it would be interesting to investigate how this algorithm performs on modern architecture in a future experiment.

2.4 BETA

Black Box Explanations through Transparent Approximations(BETA) [28] is a model-agnostic framework which aims to optimize both the "fidelity to the original model and the interpretability of the explanation" [28]. BETA constructs a small number of compact decision sets which are inherently interpretable [29] with each set attempting to capture how the Neural Network behaves at certain parts of the feature space. The framework provides reasoning as to why a specific instance was assigned their label given their feature space. This is done by ensuring that each decision set does not overlap within the feature space which they provide their decision rules for. The framework is guided by 4 properties,

Fidelity The approximations should accurately represent the behaviour of the Neural Network in all parts of the decision space.

Unambiguity A single deterministic rationale is provided for the prediction of every instance.

Interpretability The approximations constructed should be able to be understood by humans.

Interactivity The user should be able to customize the approximations based on their preference e.g. adjusting the approximation for patients within a certain age range.

BETA has been tested on a real-world depression diagnosis dataset with few features, with the majority being binary. It has not been tested on networks with complex architectures and many features. The code is propriety and has not been

made publicly available, therefore we have not considered BETA as a possible tool for comparison.

2.5 Structured Causal Models

Using the first principles of causality [30] [31] a new method of providing feature attributions for Neural Networks is introduced in the paper *Neural Network Attributions: A Causal Perspective*[32]. The approach involves viewing a Neural Network as a *Structured Causal Model (SCM)* [30] and computing the Average Causal Effect (ACE) [33] of an input neuron on a given output neuron. The standard principles of causality has made this problem tractable by finding input neurons which can be considered latently joint such as inputs which were generated by the same data-generating mechanism [32]. The proposed methodology only works on specific types of networks and the authors of the paper have considered it future work to adapt it to more generic networks. A large drawback of this methodology is that we need prior knowledge of the training dataset [32]. It has not been adapted to a generic framework and thus the code would need to be manually adapted for each new model. Adapting this to a generic framework and expanding it to be used with more complex Neural Network architectures is a possible future experiment.

2.6 Deep Visualization

The *Understanding Neural Networks Through Deep Visualization* [34] paper introduced two novel tools which aim to visualize the inner workings of a Convolutional Neural Network (CNN). The first being a tool which provides visualization into the activations produced by each layer of a CNN and the second providing visualization of the features at each layer. The downside is that these tools only work on networks that were trained with an outdated Deep Learning Framework called Caffe [35] which had it's last stable release in 2017. Lucid which is built on the Tensorflow framework provides various different forms of visualizations into Image-based models and includes both of these concepts.

2.7 DeepLIFT

Deep Learning Important Features (DeepLIFT) is a tool introduced in *Not Just a Black Box: Learning Important Features Through Propagating Activation Differences* [36] and further expanded upon in *Learning Important Features Through Propagating Activation Differences* [37] which assigns importance scores to the input variables of a model. The importance scores are assigned based on the difference from a *reference* state which is selected based on the specific problem to the *initial* state of the model. Each input is replaced by a reference value which indicates that something is lacking, an example being the presence or absence of a specific feature. Expanding this idea to Neural Networks we can assign each individual neuron a reference value which is simply the activation of the neuron given the reference input. Therefore the goal of DeepLIFT is to provide an explanation of the difference between the output of a model using its original input to the output using its referenced input. SHAP incorporates the ideas of DeepLIFT.

2.8 Pixel-wise Decomposition

In the paper *On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation* [38] the concept of pixel-wise decomposition is introduced which aims to measure how pixels positively and negatively affect the prediction of an image-based model for a particular image. Two novel methods are introduced in the paper which provides pixel-wise decomposition namely *layer-wise relevance propagation (LRP)* and a technique based on Taylor decomposition [39] which yields an approximation of layer-wise relevance propagation. Lucid also provides explanations for individual pixels in an image so for our use-case pixel-wise decomposition is redundant.

2.9 aLIME

Anchor Local Interpretable Model-Agnostic Explanations (aLIME) [40] is a system which aims to explain individual predictions using if-then rules while remaning model-agnostic. The rules provided are intuitive to humans and are usually easily

understood. aLIME achieves this by providing rules which "anchor" a prediction which means that with a high probability any other change to the instance should not have an effect on the prediction. Given a model which predicts if an adult's salary is higher or lower than $50,000 salary. An example of an anchor would be that the model always predicts the adult to earn more than $50,000 if they have beyond a high school education regardless of other features. The aim is to provide the shortest anchor which has the highest precision, however it is infeasible to exactly solve this so it does this by making use of approximations. At the moment aLIME only supports explaining individual predictions for text classifiers or classifiers that act on tables. It is an open source project so it is possible for a future contributor to adapt it to be model agnostic .

Chapter 3

Background

In this chapter we introduce the tools which we have evaluated namely *LIME*, *SHAP*, and *Lucid*. We also explain in detail how these tools provide explanations.

3.1 LIME

LIME *(Local Interpretable Model-Agnostic Explanations)* [19] is a model-agnostic explainer which attempts to provide an interpretable *explanation model* for any supervised learning problem. The overall idea of LIME is to provide an explanation for a complex model by approximating it locally with an *interpretable* one which is referred to as the *explanation model*. It is considered a *local* interpretation because LIME produces an explanation which indicates the importance of features for a single prediction and features which are locally important are not necessarily globally important, and vice versa. This is done by choosing a single instance and transforming it into a representation that LIME can understand which is explained in Section 3.1.1. Small perturbations of the transformed instance is generated which are close in proximity to the instance. The way these perturbations are generated is dependent on the type of the input and the methods include adding noise to continuous features, removing words or hiding parts of an image. This is discussed further in Section 3.1.2. The perturbed samples are then predicted by the original model. The interpretable model is trained by using the perturbed samples as the input and their corresponding predictions from the original model as their labels

which is explained in detail in Section 3.1.3. The weights from the explanation model are extracted and are considered a local explanation for the chosen instance as we are able to monitor how the change in specific features brought upon by the perturbations affect the outcome of the prediction.

A wide variety of architectures can be used when choosing an explanation model, however the more complex the explanation model is the less interpretable it becomes. The explanation model with the least complexity would be a linear model due to them being easily interpreted and for that reason it is our explanation model of choice for this thesis. The explanation model can therefore be represented by the following linear equation,

$$g(z^{'}) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i^{'} \tag{3.1}$$

where $z^{'} \epsilon \{0,1\}^M$ represents a simplified input which is explained in Section 3.1.1, M is the number of simplified input features, and $\phi_i \epsilon \mathbb{R}$ is an attribution value assigned to each feature indicating their impact on the prediction [20]. The *explanation model* is used as an approximation for the complex model and was chosen to be linear so that it is interpretable.

We can see from (3.1) that we only have a relationship between our inputs $z^{'}$ and prediction $g(z^{'})$. This means that the explanation is rather limited as we are unable to gather any information regarding the hidden layers. Suppose we have a model which takes in a list of symptoms that a patient is exhibiting and attempts to predict whether the patient has the flu or not, in Figure 3.1 we can see how LIME takes the input features which are the symptoms of the patient and provides visual feedback where each feature is given a weight which is represented by the length of the bar of the feature and a color representing how much they contributed towards or against the patient having the flu. Ultimately the visual feedback is given to a Doctor as a reference and they should be the one to give the final diagnosis.

3.1.1 Internal data representation

Before the model can be explained, LIME performs a transformation on the input to create it's own internal representation that it can interpret. The transformation

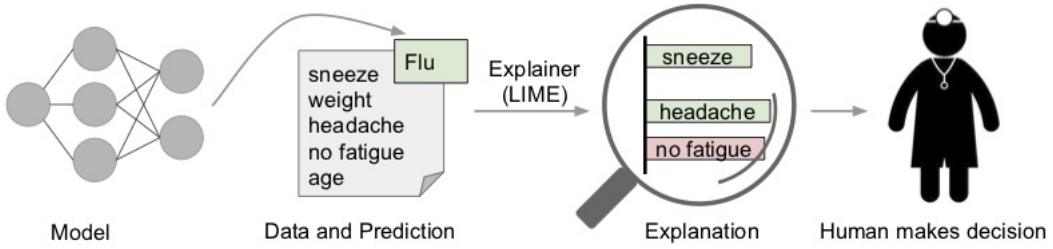


Figure 3.1: Example of LIME on a model that predicts whether a patient has the flu. **Image from** [19].

is done behind the scenes and is dependent on the type of the input data.

- For text data a model would commonly use complex representations such as word embeddings but a simpler representation is needed such as a binary vector where 1 indicates the presence and 0 the absence of a given word. This can be formally expressed as, $X' = \{0, 1\}^p$ where p is the number of words in the instance being explained. For example given a corpus of words {The, cat, dog, jumped, leaped, over, under, the, fence} and the input sentence "The dog jumped over the fence", we could represent the input with the binary vector {1, 0, 1, 1, 0, 1, 0, 1, 1}.

- For images a binary vector is used to indicate the presence or absence of super pixels [41] which is a grouping of similar pixels in the image, where 1 indicates that the original value of the super pixel is used and 0 the pixel is set to grey which represents missing. This can be formally expressed as, $X' = \{0, 1\}$ where p is the number of identified super pixels in the input image. Identifying super pixels is usually done by image segmentation algorithms such as Quickshift [42].

- For tabular data such as matrices it is dependent on the type of data. In the case of numerical data the representation is simple enough so the identity mapping is used $X' = X$. For categorical data a binary vector is once again used which can be formally expressed as $X' = \{0, 1\}^p$ where p is the number of features used by the model. In the binary vector, 1 indicates that the feature takes on the original category and 0 indicates that the feature is mapped to a different category sampled according to the distribution of the training

data. Categorical data, is data which is grouped into some sort of category or multiple categories. For example, a categorical feature would be used to represent the type of an animal which could be either a cat, dog, or mouse. In this case we would have 3 categories and we could use a 0 to present the category cat, 1 to represent the category dog, and 2 to represent the category mouse.

Regardless of the data type, LIME creates a mapping function which transforms the internal data representation back into its original form for use when predicting on the actual model.

3.1.2 Sampling from the input

Once the input has been transformed into a internal representation, LIME begins sampling from the new input in order to create perturbed samples. The sampling process like the data transformation process is dependent on the type of input data.

- For text data, the sampling process involves removing uniformly at random, words from the instance. In the previous example "The dog jumped over the fence", some permutations that could be created from sampling would be "dog jumped over the fence", "over the fence", or "jumped".
- For images, the sampling process is as simple as setting values chosen uniformly at random in the vector to 0 therefore setting them to missing.
- For tabular data such as matrices, it is once again dependant on whether the data is numerical or categorical. For categorical features we set them to random values within their category. For numerical features, the instance is perturbed by sampling from a normal distribution and doing the inverse operation of mean-centering and scaling.

3.1.3 Training the explanation model

Finally the explanation model has to be trained, given a prediction instance x, i.e. a specific feature set that predicts a specific probability under the model, after LIME

has transformed the input into a internal data representation $x^{'}$ and a specified number of perturbed samples have been generated. An individual perturbed sample z is then reconstructed to represent an actual sample if needed. This is mainly done by setting the missing features to 0. The reconstructed input $z^{'}$ is predicted on the model and LIME records how the new prediction probabilities have changed. This is done for each perturbed sample, the more perturbed samples we use the more accurate the results are. By doing this LIME is able to explain the contribution of each feature towards the prediction. We choose the *explanation model* by using the following equation,

$$\xi(x) = \underset{g \epsilon G}{\operatorname{argmin}} L(f, g, \pi_x) + \Omega(g) \tag{3.2}$$

where f is the model that we are trying to explain, g is all potential explanation models, $\xi(x)$ represents the optimal explanation model, $L(f, g, \pi_x)$ is the approximation of the difference between f and g, π_x is a proximity measure between an instance z to x to define locality around x where instances closer to x are given a higher weighting, and $\Omega(g)$ is a complexity score given to the current explanation model which needs to minimized. LIME defaults to linear sparse models which are given a complexity rating $\Omega(g) = 0$ as the domain for possible explanation models such as in (3.1). We can see in Figure 3.2 the linear decision boundary that LIME has learnt within a complex space. The crosses represent the perturbed samples which were generated training the explanation model, where the bold cross is the initial input. The complex space is the models decision function and the dotted line is the linear explanation model that LIME has learnt.

3.1.4 Providing feedback to the user

The end goal of LIME is to provide textual and visual artifacts to the end user that explains the chosen prediction. The *explanation model* itself is used as the explanation for the chosen prediction. The feature attributes ϕ_i are presented to the user in a representation that is interpretable by humans.

- For feedback on an image-based model, suppose we had a model which attempts to predict whether a given image is of an *Electric guitar*, *Acoustic*

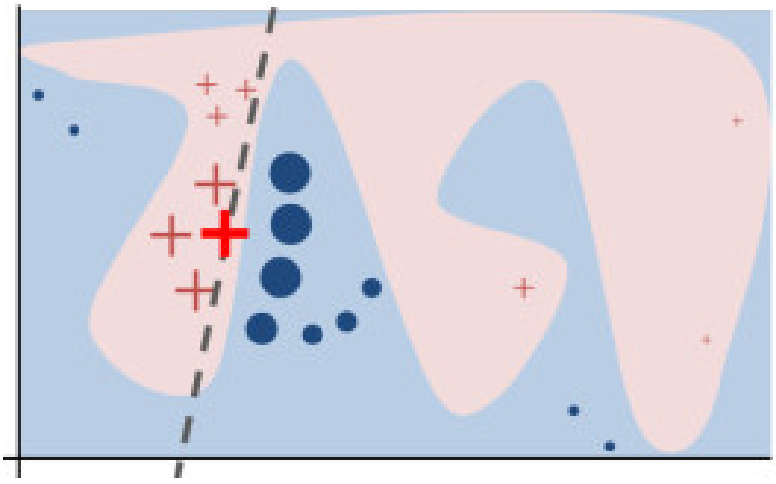


Figure 3.2: Linear decision function learnt in complex space. **Image from** [19].

guitar, or *Labrador*. In Figure 3.3, for the 3 respective classes the relevant super pixels which had a positive impact on the prediction of that particular class are displayed and the rest of the image is greyed out.

- In the case of a model trained on text data, suppose we had a model which given an article attempts to predict whether the topic is about *Atheism* or *Christianity*. In Figure 3.4 we can see the prediction probabilities of each class and the words which attributed towards the prediction are highlighted and given a weighting.

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 3.3: Explanation provided by LIME on an image-based model. **Image from** [19].

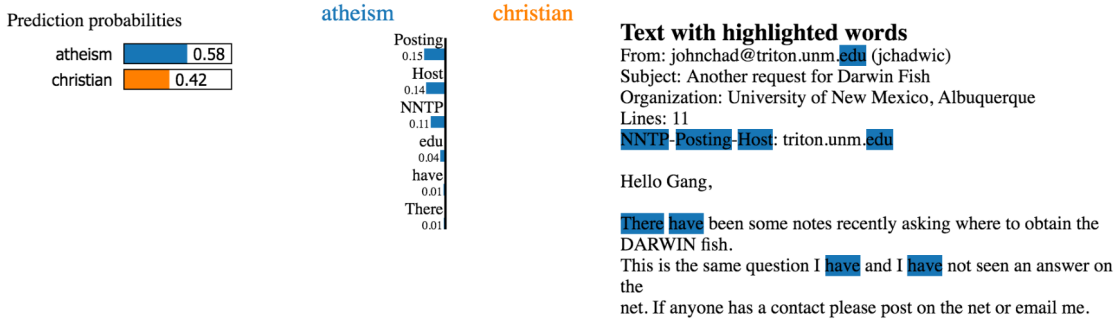


Figure 3.4: Explanation provided by LIME on an text-based model. **Image from** [43].

3.1.5 SP-LIME

Since LIME is only able to provide an explanation for a single prediction at a time, it is up to the user to explain multiple instances in order to gain a global understanding of the model. A tool called *Submodular Pick LIME (SP-LIME)* [19] can be used to help the user choose instances that do not overlap and each extra instance should bring a better understanding of the model on a global level. A human is only willing to view a certain number of explanations in order to understand a model and this is denoted as B instances. A *pick step* is provided which selects the optimal B instances for the user to inspect. The pick step aims to pick a diverse set of explanations that are non-redundant to show the user in order to explain how the model behaves globally. We denote W as being an *explanation matrix* that represents the local importance of the interpretable components for a set of instances $X(|X| = n)$ [19]. Each row in W denoted as i represents an instance and each column denoted as j represents each feature or component. We denote I_j as the global importance of that component in the explanation space. "Intuitively, we want I such that features that explain many different instances have higher importance scores" [19].

An example of how the pick algorithm operates can be seen in Figure 3.5. The rows represent instances and the columns are features. The goal is to choose new instances so that new features are being explored. If the second row was one of the picked explanations then there would be no reason to chose the third row as they both explain the same set of features $f2$ and $f3$. The fifth row would be a

good instance to pick as it explains a distinct set of features $f4$ and $f5$. Using our importance function I it should score feature $f2$ higher than feature $f1$, i.e. $I_2 > I_1$ since $f2$ is used to explain more instances. This non-redundant coverage intuition can be formally expressed as,

$$c(V, W, I) = \sum_{j=1}^{d'} \mathbb{1}_{[\exists i \epsilon V : W_{ij} > 0]} I_j \tag{3.3}$$

"where we define coverage as the set function c that, given W and I, computes the total importance of the features that appear in at least one instance in a set V" [19]. The pick problem can be defined as,

$$\text{Pick}(W, I) = \underset{V, |V| \leq B}{\text{argmax}}\, c(V, W, I) \tag{3.4}$$

consisting of finding the set $V, |V| \leq B$ that achieves the highest coverage. This algorithm is known as *submodular pick* and is a NP-hard problem [44].

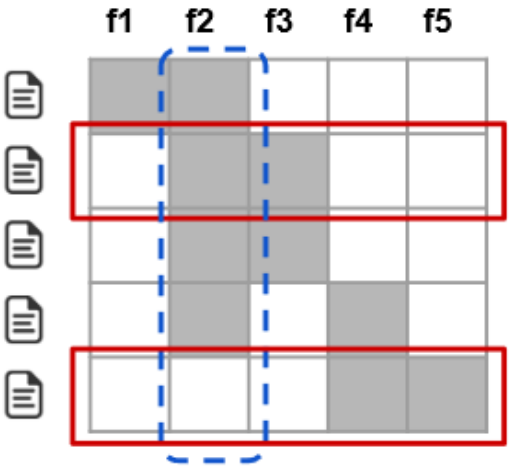


Figure 3.5: "Toy example W. Rows represent instances (documents) and columns represent features(words). Feature f2 (dotted blue) has the highest importance. Rows 2 and 5 (in red) would be selected by the pick procedure, covering all but feature f1" [19]. **Image from** [19].

3.1.6 Limitations

LIME works as well as the explanation model that is built, if the explanation model does not approximate the real model well then LIME will not perform well in turn. Since LIME attempts to provide visual and textual artifacts to the user for interpretation, if there are many features such as in the thousands then even if the interpretation is accurate, a human will not be able to interpret so much information. LIME is also only able to interpret a single prediction at a time, this may not be representative of how the model performs on a global scale so it is up to the user to properly make use of this in order to gain a global understanding of their model.

3.2 SHAP

SHAP(Shapley Additive exPlanations) [20] is a tool which combines and adapts various different interpretation tools with LIME being one of them. It can be shown that these different tools all attempt to create the same explanation model in the form of (3.1) and are referred to as *additive feature attribution methods* [20]. These tools were all created in isolation and there was no simple way to determine when one method was preferable to another. It can be shown that there is a single unique and desirable solution between these different *additive feature attribution methods* that adheres to three desirable properties derived from Shapley value estimation methods [2] [45] [46]. These properties are *local accuracy*, *missingness*, and *consistency* which we will further discuss in Section 3.2.1. From this unique solution, *SHAP values* [20] are derived which can be seen as a unified measure of feature importance.

3.2.1 Properties

Local Accuracy

Requires the *explanation model* g to at least match the output of the original model f for the simplified input $x^{'}$ (which corresponds to the original input x),

$$f(x) = g(x^{'}) = \phi_0 + \sum_{i=1}^{M} \phi_i x_i^{'} \tag{3.5}$$

where M is the number of simplified inputs and ϕ_i are their values [20].

Missingness

Constrains features where $x_i^{'} = 0$ to have no attributed impact [20].

$$x_i^{'} = 0 \implies \phi_i = 0 \tag{3.6}$$

Consistency

If a model changes so that some simplified input's contribution increases or stays the same regardless of the other inputs, that input's attribution should not decrease. Let $h_x(x^{'})$ be a mapping function that transforms a simplified input $x^{'}$ into the original input x, $f_x(z^{'}) = f(h_x(z^{'}))$ and $z^{'} \setminus i$ denote setting $z_i^{'} = 0$. For any two models f and $f^{'}$, if

$$f_x^{'}(z^{'}) - f_x^{'}(z^{'} \setminus i) \geq f_x(z^{'}) - f_x(z^{'} \setminus i) \tag{3.7}$$

for all inputs $z^{'} \epsilon \{0,1\}^M$, then $\phi_i(f^{'}, x) \geq \phi_i(f, x)$ [20].

3.2.2 Unique solution

There exists a single unique solution which can be written as an *Additive feature attribution method* that adheres to all 3 three properties.

$$\phi_i(f, x) = \sum_{z^{'} \subseteq x^{'}} \frac{|z^{'}|!(M - |z^{'}| - 1)!}{M!} [f_x(z^{'}) - f_x(z^{'} \setminus i)] \tag{3.8}$$

where $|z^{'}|$ is the number of non-zero entries in $z^{'}$, and $z^{'} \subseteq x^{'}$ represents all $z^{'}$ vectors where the non-zero entries are a subset of the non-zero entries found in $x^{'}$. This equation was derived from cooperative game theory results, where the values ϕ_i are referred to as Shapley values [47]. It has been proven that Shapley values adhere to properties 1 and 3, but not 2. The previous *additive feature attribution methods* such as LIME are proven to sometimes violate properties 1 and 3, but adhere to property 2. Thus the previous *additive feature attribution methods* are adapted to use Shapley values so that they adhere to all 3 properties and therefore is the solution to (3.8). We discuss how these techniques are adapted to use Shapley values in Section 3.2.4. A new unified measure of feature importance is proposed called *SHAP Values* [20] which is the solution to (3.8) and is used to adapt previous *attribution methods*

3.2.3 SHAP Values

SHAP Values (Shapley Additive exPlanation) are the Shapley values of a conditional expectation function of the original model, which means they are the solution to (3.8), where $f_x(z^{'}) = f(h_x(z^{'}))) = E[f(z)|z_S]$, and S is the set of non-zero indexes in $z^{'}$ [20]. Although the exact computation of SHAP values are challenging, by combining insights gathered from current *additive feature attribution methods* it is possible to approximate them. SHAP makes two other assumptions *model linearity* and *feature Independence* in order to further simply the expected values,

$$f(h_x(z^{'})) = E[f(z)|z_S],$$

applying our two assumptions we can simplify this as,

$$\approx f([z_S, E[z_{\overline{S}}]]) \tag{3.9}$$

"SHAP values attribute to each feature the change in the expected model prediction when conditioning on that feature. They explain how to get from the base value $E[f(z)]$ that would be predicted if we did not know any features to the current output $f(x)$" [20]. Figure 3.6 shows a single ordering. In the case of features which are dependant on one another or a nonlinear model, the order of which the

features are added to the expectation is important. In that case the SHAP values are calculated by averaging the ϕ_i values across all possible orderings [20].

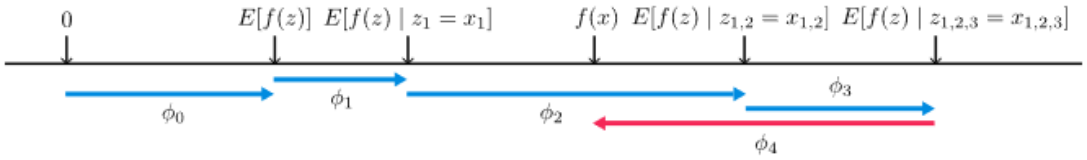


Figure 3.6: How the SHAP values are computed. **Image from** [20]

3.2.4 Types of explainers

Using SHAP Values, the previous *additive feature attribution methods* can be adapted. There are a total of 6 tools which are adapted, however in this thesis we only look at the adaptions of LIME and DeepLIFT[37][36].

Kernel SHAP

Adapting LIME to adhere to properties 1 and 3 which it initially violates gives rise to *Kernel SHAP*, which makes use of SHAP Values. It remains model agnostic. If we refer back to (3.2), we recall that $\Omega(g)$ is the term which penalizes the *explanation model* based on how complex it is, $\pi_{x'}$ is the locality around input $x^{'}$, and $L(f, g, \pi_{x'})$ is the loss function which indicates the difference between the original model f and the explanation model g. These three values are determined heuristically by LIME. However SHAP [20] provides methods for choosing them that adheres to all three properties and has been named *Shapley Kernel*. Under (3.1), the specific forms of $\pi_{x'}$, L, and Ω which make (3.2) consistent with the three properties of SHAP are:

$$\Omega(g) = 0,$$

$$\pi_{x'}(z^{'}) = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)},$$

$$L(f, g, \pi_{x'}) = \sum_{z' \epsilon Z} [f(h_x(z')) - g(z')]^2 \pi_{x'}(z'),$$

where $|z'|$ is the number of non-zero elements in z'. $\Omega(g)$ remains 0 because we still use linear models, $\pi_{x'}(z')$ is the weighting of how close the sampled input z' is to the simplified input x' where M is the number of features in x', and $L(f, g, \pi_{x'})$ is the mean squared error which depends on the weighting of $\pi_{x'}(z')$.

Deep SHAP

DeepLIFT was proposed as a recursive prediction explanation model for deep learning [36][37]. The user assigns each feature a reference value which is typically a background feature for that particular feature. For each input x_i a value $C_{\Delta x_i \Delta y}$ is given which represents the effect of setting that particular input to it's reference value. Once again we introduce the mapping function $x = h_x(x')$ that converts a simplified binary input x' to the original input x, where 1 indicates the input takes it's original value and 0 it takes it's reference value. DeepLIFT therefore uses the following property named the *summation-to-delta property* [20],

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta o} = \Delta o, \tag{3.10}$$

where $o = f(x)$, $\Delta o = f(x) - f(r)$, $\Delta x_i = x_i - x_{r_i}$, and r is the reference input. (3.10) can be written in the form of (3.1) by setting $\phi_i = C_{\Delta x_i \Delta o}$ and $\phi_0 = f(r)$, therefore it is an *additive feature attribution method*. Although *Kernel SHAP* is completely model-agnostic, we can leverage extra knowledge about deep networks in order to increase computational performance. DeepLIFT by itself only satisfies the *local accuracy* and *missingness* properties, but if we interpret the reference values r_i as $E[x]$ in (3.9) then DeepLIFT approximates SHAP values and *consistency* is also adhered to given that the model is linear and the features are independent of one another. The new method is labeled *Deep SHAP* which not only computes SHAP Values but also combines the "SHAP values computed for smaller components of the network into SHAP values for the whole network" [20].

3.2.5 Limitations

SHAP has a similar limitation to LIME which is that it only uses the input and output of a model for interpretation and therefore no knowledge of the hidden layers are gained. SHAP also assumes *model linearity* and *feature independence* which is an assumption which more complex models may not adhere to. SHAP is also more computationally intensive than LIME due to extra constraints and the need to approximate Shapley values.

3.3 Lucid

"Lucid is a collection of infrastructure and tools for research in Neural Network interpretability" [21]. Several powerful techniques such as *feature visualization*, *attribution*, and *dimensionality reduction* have been studied for interpreting Neural Networks, however these techniques have been researched in isolation and there has been not been research on how these techniques may complement one another. Lucid is primarily focused on interpretability into image-based networks such as CNNs where various parts of Lucid can be used in order to gain some interpretation into the hidden layers, the previously isolated interpretation techniques have been unified to create a rich interface [48]. The interface that Lucid provides can be seen in Figure 3.7, the *layers* are the layers which we are able to gain insight into, *atoms* are the various ways in which we can group the pixels in the input images which will be explained further in Section 3.3.2, *content* is the type of information we can gain, and *presentation* is how this information is relayed to the user [48].

3.3.1 Semantic dictionaries

The previous two tools *LIME* and *SHAP* only attempt to provide interpretation between input and output layers, where as Lucid provides interpretation between any two layers. Each layer in a neural network can be seen as a three-dimensional cube where each cell represents an *activation*, which is the amount that a specific neuron fires. The x- and y-axes represent the positions in the image, and the z-axes the particular channel. Usually activations are represented as abstract vectors that are not interpretable. A *Semantic dictionary* is introduced which provides a more

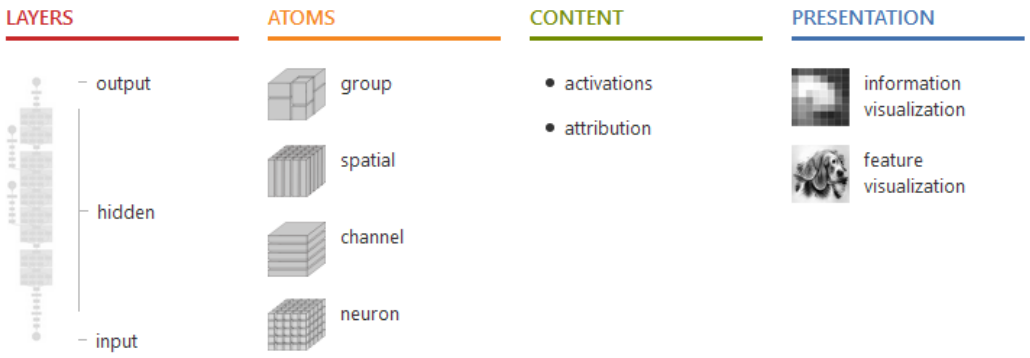


Figure 3.7: The interface that Lucid provides. **Image from** [48].

meaningful human interpretable representation of activations. *Semantic Dictionaries* attempt to provide each activation with a *canonical example* by mapping each activation with a visualization of that particular neuron sorted by the magnitude [48]. This allows activations to map to similar human representations such as a "Floppy ear", or "Dog snout" when interpreting a model trained on classifying dogs. There are many ways to approximate the relationship between features and how they are used to arrive at a prediction however in this case they are linearly approximated.

3.3.2 Types of Activations

Since we can visualize the group of neurons as a cube, there are different ways we can slice this cube into groupings to form activations. The way each activation breaks up the cube can be seen in Figure 3.8.

Individual Neurons

A *Saliency map* [49] is a simple heat map which highlights pixels of an input image that cause the most output classification. Since this method works with each individual neuron we run into two big problems, each pixel in an image is usually entangled with surrounding pixels and is heavily affected by transformations such as contrast or brightness changes. The second problem is that *saliency maps* are

fairly limited and do not allow multiple class attributions to be displayed at one time.

Spatial Activations

Due to the limitations of the previous approach only being able to show the attributions to a single class at a time, a different approach is introduced called *Spatial activations* [48]. Spatial activations are performed from a chosen layer to all *n*-output classes. Dimensionality reduction is then performed on this n-dimensional vector to produce a multi-directional *saliency map*. Once this *saliency map* is overlayed on the magnitude-sized activation grid we obtain an *information scent* over this attribution space. This allows attribution between layers, however different concepts are being detected together and if we continue to use spatial positions these concepts will remain entangled.

Channel Activations

A *channel* can be seen as a specific layer or view of an image. There are detectors at each channel representing a different view of the image [48]. For a normal color image we have the width and height to represent the pixels and the channels are the 3 color channels. In more complex representations of images these channels can be specialized to detect certain features such as eyes or noses. By using spatial activations, the attribution is aggregated over all channels. which means we can not tell which detectors at each position most contributed to the final output classification. We can slice the cube by channels rather than spatial locations, which allows us to tell which detectors contributed the most to the output. In the case of a model which attempts to classify dogs, detectors may include those which detect "Floppy ears" or "Dog snouts".

Neuron Groups

Individual neurons can provide the most information, however for larger networks where there may be tens of thousand of neurons this is too much information for humans. *Spatial activations* and *Channel activations* both provide different information about the layers, however we would gain the most if we could combine both

of these techniques. There is a field of research, called *matrix factorization* which studies optimal ways of breaking up matrices, we can make use of these methods by flattening the cube into a matrix of spatial locations and channels. Applying *matrix factorization* on this newly created matrix, we can obtain more meaningful explanations of the layers. By using the techniques in *matrix factorization* we are able to choose what we prioritize in the activations such as describing the gradient, fully describing the activations, or if we want to fully describe the attributions. The problem with this method is that the *neuron groupings* [48] we use for a particular image can not be used for another image as each image would require a unique set of groups.

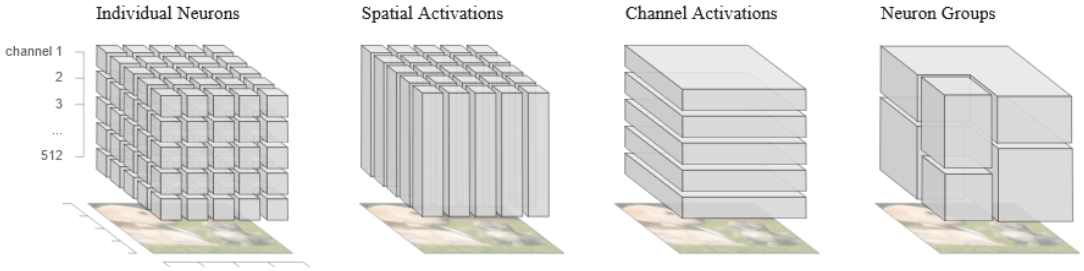


Figure 3.8: Different types of activations in Lucid. **Image from** [48].

3.3.3 Limitations

The largest limitation is that Lucid currently only provides explanations into image-based Neural Networks and further research is being done to extend this to other architectures. Lucid is only built to be used with Tensorflow and it currently does not support the newer Tensorflow 2.0. Lucid is built on a volunteer basis and therefore there is not a lot of support on how to adapt the techniques to other domains. Since Lucid requires extensive knowledge about Neural Networks and interprets hidden layers, it is useful for the user who is creating the model but is not useful to the end user who has no knowledge about the architecture.

Chapter 4

Evaluation

In this chapter we use LIME, SHAP and Lucid (for image-based models) on various types of models to gain an understanding on how predictions on different model architectures are explained.

4.1 House Prices

We start off by training a Neural Network for predicting house prices as introduced in Section 1.1. We make use of the Boston Housing Dataset [50] "which contains US census data concerning houses in various areas around the city of Boston. Each sample corresponds to a unique area and has about a dozen measures" [51]. The dataset provides information such as Crime (CRIM), areas of non-retail business in the town (INDUS), the age of people who own the house (AGE). The dataset contains 506 samples where the target variable is the median value of owner-occupied homes in $1000's (MEDV).

4.1.1 Model Architecture

We train a simple Neural Network with only an input and output layer in order to minimize the complexity which can be seen in Figure 4.1. Our Neural Network takes in the 13 input features and outputs the predicted median price. Due to this being a regression problem we use *Mean Absolute Error (MAE)* as our loss function

which is the arithmetic average of the absolute error between our predicted value and the true value [52].

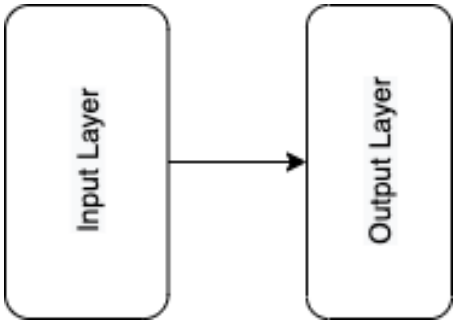


Figure 4.1: Houses Prices model architecture

- Optimizer = Adam
- Learning Rate = 0.01
- Loss Function = Mean Absolute Error
- Training Loss = 86
- Test Loss = 76

4.1.2 LIME

Figure 4.2 is the explanation LIME provides for an area in Boston where the median price of housing was predicted to be $19870. From this figure we are able to deduce that the feature that had the largest negative impact on the median price of this area is the % lower status of the population (LSTAT) which is the proportion of the population that is considered lower status in terms of either education or type of employment and can be measured as *1/2 (proportion of adults without, some high school education and proportion of male workers classified as laborers)* [53] and the feature which had the largest positive impact is the average number of rooms per dwelling (RM).

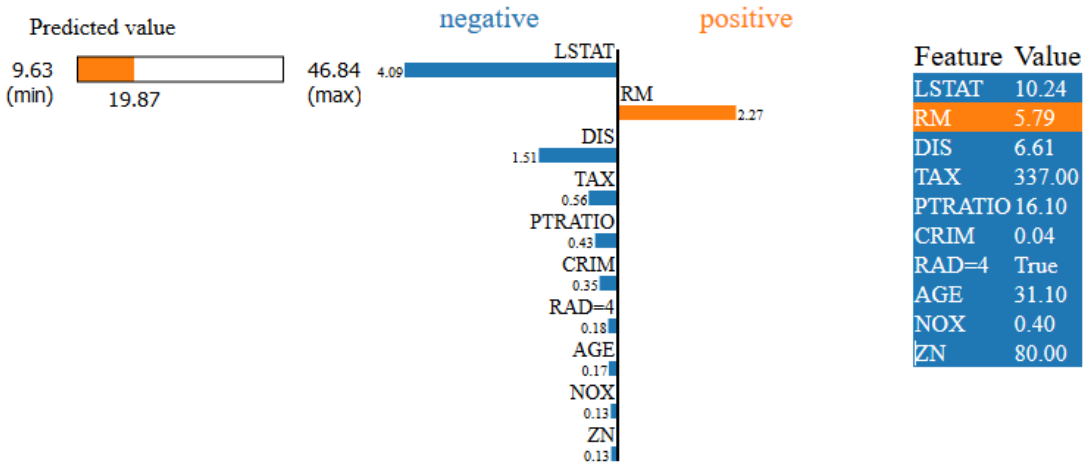


Figure 4.2: LIME explanation for the predicted median price for an area in Boston.

4.1.3 SHAP

Figure 4.3 is the explanation SHAP provides for the same Boston area used in LIME. The base value in the figure is shown to be 22.89 which is the average predicted value output over the entire training dataset. This can also be regarded as the models output given no input features. The figure highlights how each feature contributes to pushing the prediction from the base value. The red arrows are pushing towards the prediction which indicates that they increase the price and therefore have a positive impact. The blue arrows push against the prediction which indicates that they decrease the price and therefore have a negative impact.

The explanation provided is a lot different to that of LIME's, because SHAP indicates how each feature either increases or decreases the prediction starting at the base value. An obvious difference between this and the SHAP explanation is that RM seems to have a negative impact on the price. This is due to the average RM being 6.28 in the dataset and since this is below the average of 5.787 it has an overall negative impact. Figure 4.4 is the explanation that SHAP provides over multiple different areas. This is known as a summary plot and it compares SHAP values of multiple instances and how their impact on the model prediction is related to the magnitude of that feature. Features which have larger SHAP values increased the median price where as those with smaller SHAP values decreased the

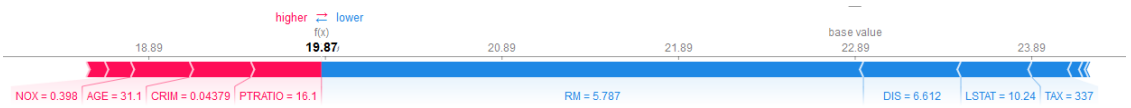


Figure 4.3: SHAP explanation for the predicted median price for an area in Boston.

median price.

Figure 4.4: SHAP explanation over all the areas in Boston.

4.1.4 Comparison

Comparing LIME in Figure 4.2 and SHAP in Figure 4.3 which are the explanations of individual predictions they are largely similar. Both provide quantitative values

for the effect each feature had on the prediction. SHAP, however explains how each feature affects the models average output given no input features. The largest advantage of SHAP is that the provided SHAP values are averaged over multiple instances known as background samples. SHAP is also able to easily visualize the SHAP values of multiple instances at a time as seen in Figure 4.4.

4.2 MNIST

The MNIST digit classification problem is one of the most popular machine learning problems in image processing. The dataset consists of *60000* training images and *10000* test images of handwritten digits with a size of 28×28. The goal is to determine what digit class *0-9* the specific image belongs to.

4.2.1 Model Architecture

For this problem we trained a Convolutional Neural Network (CNN). The architecture can be seen in Figure 4.5. *Convolutional Layers* are where features are extracted from our image, *Max Pooling Layer* takes groups of values and changes them all to the maximum value among them, *Dropout Layers* drop random weights between layers in order to prevent over-fitting, the *Flatten layer* converts a matrix into a single flat array. The input is the image 28×28 and the output contains 10 values, each being a probability that the image belongs to a specific digit class.

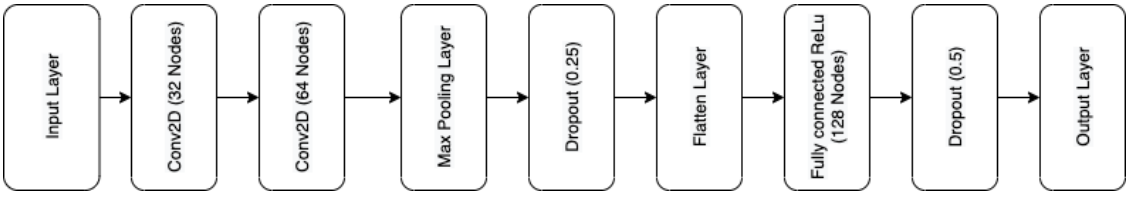


Figure 4.5: Mnist model architecture.

- Optimizer = Adadelta
- Learning Rate = 0.001

- Loss Function = Categorical Cross-Entropy
- Training Accuracy = 91.32%
- Test Accuracy = 91.95%

4.2.2 LIME

Since there are $28 \times 28 = 784$ potential features, LIME creates a perturbed input z' by randomly choosing subsets of the non-zero features. We limited the amount of perturbed samples generated by LIME to 1000 in order to increase the speed of the computation. In Figure 4.6 we can see for the specific handwritten digit *5*, the green pixels represents pixels which attributed towards predicting that the digit is 5 and the red pixels attributed towards the digit being 3.

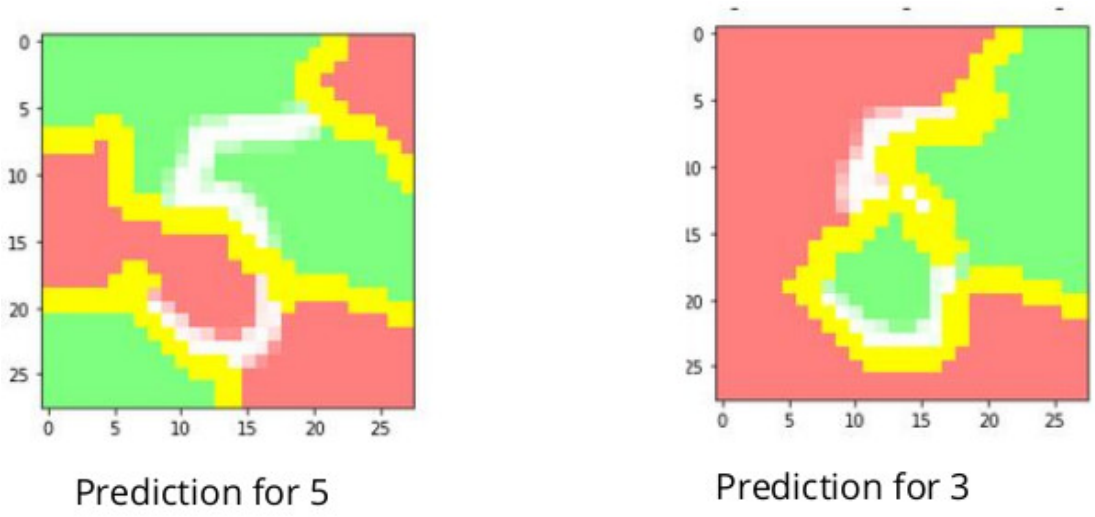


Figure 4.6: Example of LIME visual feedback for the digit 5.

4.2.3 SHAP

When making use of SHAP we are able to take an expectation of multiple samples which are referred to as *background samples* in order to gain a more accurate representation of the attributed SHAP values. For this problem we took a set of

1000 background samples. In Figure 4.7 we can see the SHAP explanation for the digit 3. Since our model provides a probability that the digit belongs to it for each class we are able to see which parts of the image added to that probability. The model gave the class 3 the highest probability with 34.701%. From the red pixels we can see that it mostly looked at the curves in the digit and the middle line to predict that it was the digit 3. From the blue pixels we can see that the inward curls at the end of the top and bottom curves decreased the probability. The second highest probability was the digit 2 with 21.261% and once again we are able to see which pixels increased this probability and which decreased it. We have another example in Figure 4.8 for the digit 5. Our model is much more certain in this example as we can see the probability of it belonging to class 5 is 64.285%. The red pixels also look a lot more intuitive as we can see that the model looked at the the top horizontal and vertical line, as well as the empty space below it to conclude that this is the digit 5.

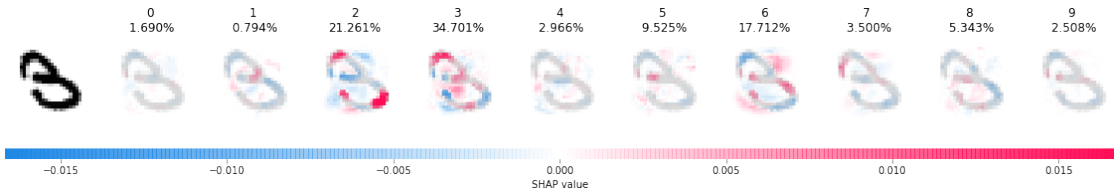


Figure 4.7: SHAP values plotted for digit 3.

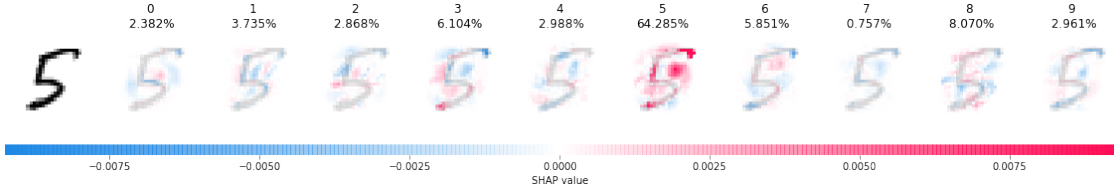


Figure 4.8: SHAP values plotted for digit 5.

4.2.4 Lucid

For Lucid we look at the spatial activations into two of the hidden convolutional layers labeled *Conv1* and *Conv2* respectively. Lucid provides a user interactive interface for viewing the activations, however in this example we will showcase static visual feedback. In Figure 4.9 we showcase the digit 3 and in Figure 4.10 the digit 5. For the two hidden layers *Conv1* and *Conv2* we are able to use spatial activations to see where and how much a specific set of pixels in the Conv1 Layer attributed in the target layer Conv2. The orange square is the set of pixels we are looking at in the starting layer and the whitened pixels in the target layer is where the activations happen

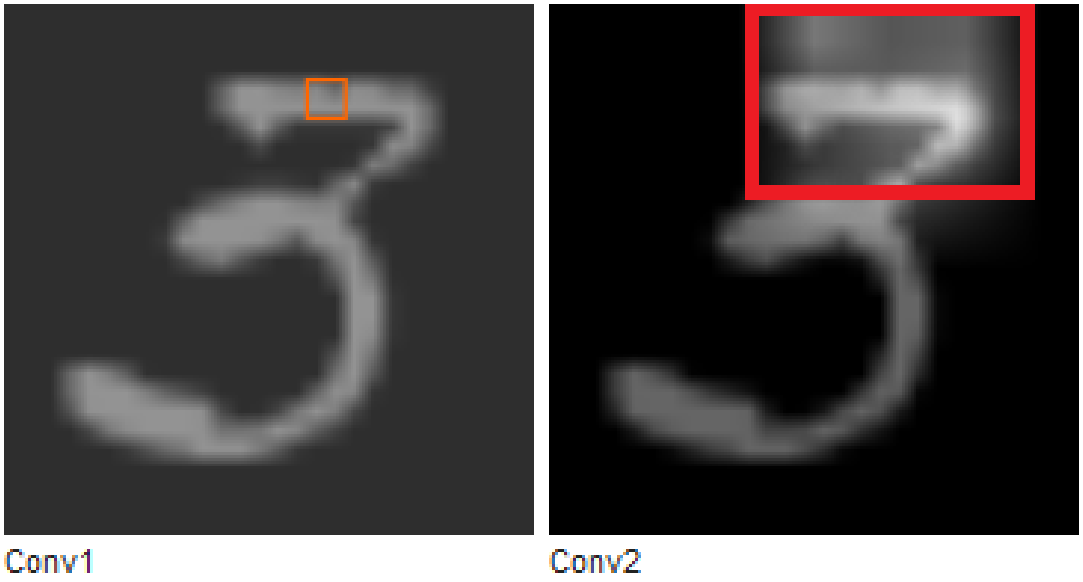


Figure 4.9: Lucid spatial activation for the digit 3.

4.2.5 Comparison

Looking at the explanation that LIME produced in Figure 4.6 it is quite clear which set of pixels had a positive and negative contribution for both the classes 5 and 3. Although it is easy to understand, it may be too simple, since by making use of super pixels we can't look at small sets or individual pixels but only large

Figure 4.10: Lucid spatial activation for the digit 5.

segments. SHAP on the other hand marks individual pixels which provides much more information into exactly which parts of the image attributed positively to which class. SHAP also provides the contribution that each pixel had with regards to the prediction which is the SHAP value magnitude as indicated by hue of the pixels. Lucid provides a completely different explanation. Rather then how the pixels contributed to each class, Lucid is able to explain between 2 layers which pixels from the source layer was responsible for activations in the destination layer. This provides intuition into how different layers interact with each other and can be valuable when attempting to understand the deeper inner workings of the network.

4.3 Cats vs Dogs

The Cats vs. Dogs is an image classification problem of finding suitable detectors for differentiating whether an image is of a cat or a dog. The dataset consists of 1000 *training images* and 500 *test images* with a size of $150 \times 150 \times 3$.

4.3.1 Model Architecture

We also trained a CNN for this model and it's architecture can be seen in Figure 4.11 which is similar to that of the MNIST problem in Section 4.2.1. The input is the image matrix $150 \times 150 \times 3$ and the output is the probabilities of the image being a cat or dog.

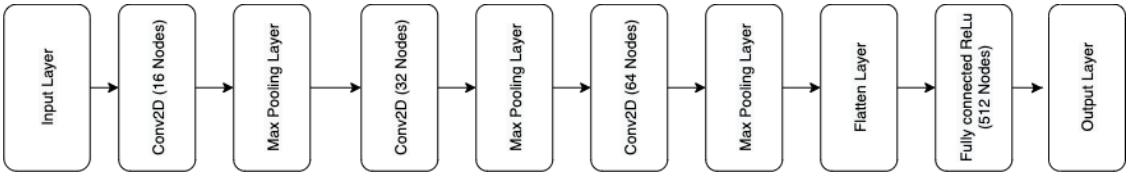


Figure 4.11: Cats and Dogs model architecture.

- Optimizer = RMSprop
- Learning Rate = 0.001
- Loss Function = Binary Cross-Entropy
- Training Accuracy = 99.1%
- Test Accuracy = 73.7%

4.3.2 LIME

In Figure 4.12 we have an image of a cat and we visualize which super pixels attributed towards the **prediction of the image being a cat**, the green sections represent pixels that attributed positively towards it being a cat where as the red sections represent pixels which attributed positively towards it being a dog. As a human this seems to make some intuitive sense, the ears of the cat are marked as positive attributions due to the fact that we associate dogs with "droopy ears", where as the cat has "fluffy ears". Another example is in Figure 4.13 where our model **incorrectly identified the cat as a dog**, the image believes the ears don't belong to a dog since it can be seen as "fluffy ears", however it believes that a large segment of the face belongs to a dog and overall it seems to be a stronger predictor than the ears and therefore the image was ultimately predicted to be a dog.

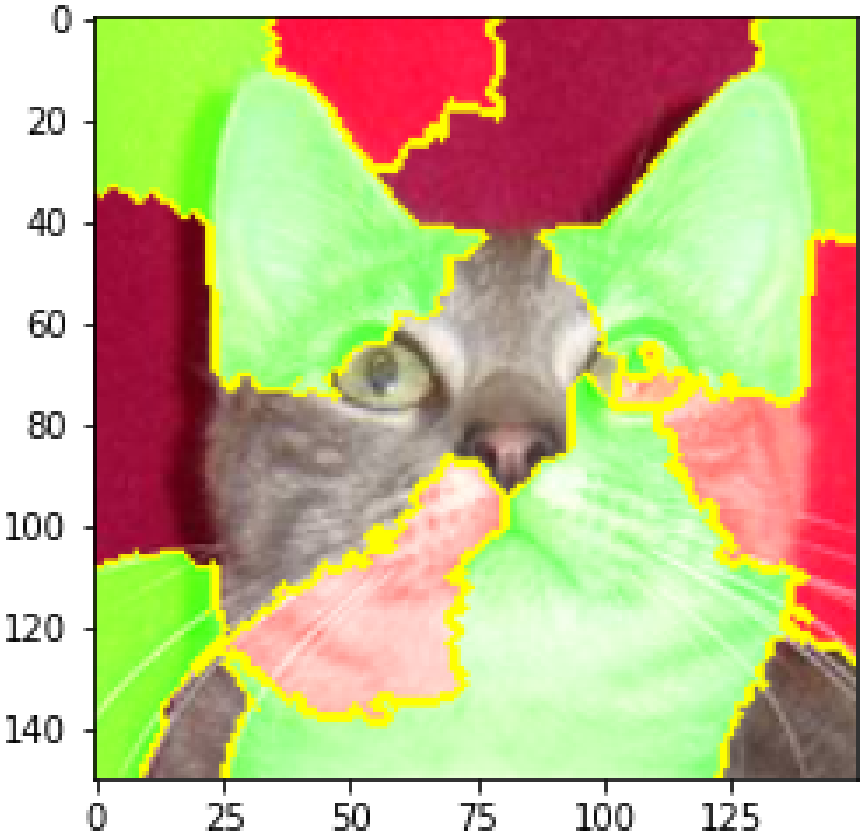


Figure 4.12: LIME explanation for the prediction of a cat.

4.3.3 SHAP

In Figure 4.14 we show the SHAP values of 5 input samples and how they attributed towards whether the image is that of a cat or dog. SHAP provides much more detailed descriptions as each individual pixel is marked but LIME seems to be easier to understand as super-pixels are marked rather than individual ones. In Figure 4.14 we can see the SHAP values for 5 samples. The middle column are the SHAP values which attributed towards being a cat, where as the right column are the SHAP values which attributed towards being a dog. Red indicates a positive attribution, where as blue indicates a negative attribution. Alongside highlighting individual pixels SHAP also provides their prediction strength represented by their SHAP values.

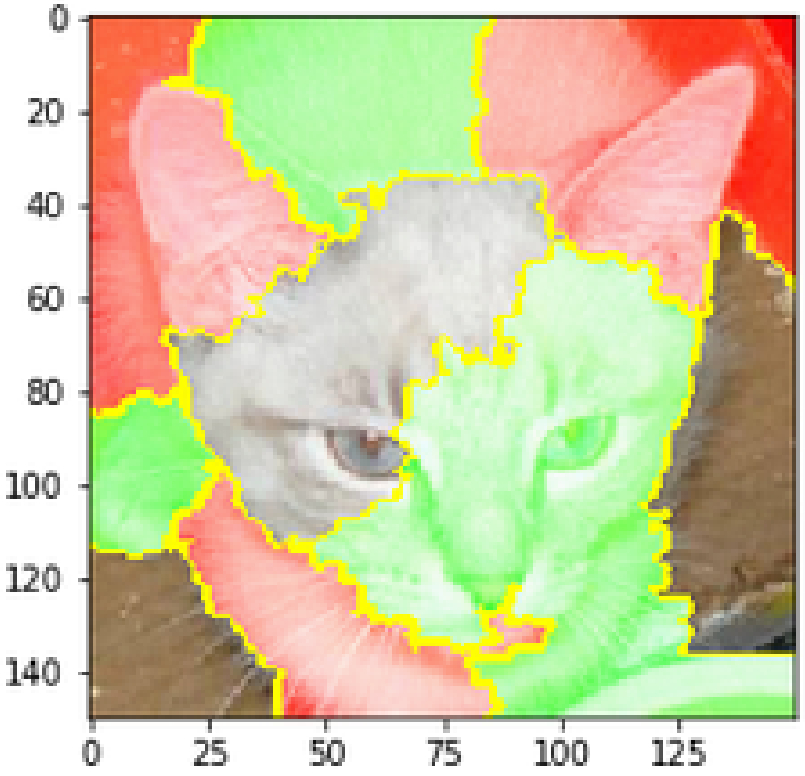


Figure 4.13: LIME explanation on an image of a cat which was falsely predicted to be a dog.

4.3.4 Lucid

In Figure 4.15 we once again look at spatial activations between two hidden convolutional layers. However, in this case we included the gradient in the starting layer which shows the magnitude of the activations from the previous layers. In Figure 4.15 for the two hidden layers *Conv2* and *Conv3* we are able to use spatial activations to see where and how much a specific set of pixels in the Conv2 Layer attributed in the target layer Conv3. The orange square is the set of pixels we are looking at in the starting layer and the whitened pixels in the target layer is where the activations happen.

Figure 4.14: Shap values for 5 different pictures of cats and dogs.

Figure 4.15: Lucid spatial activations between two layers in an image of a cat.

4.3.5 Comparison

Since this is also a image-based model with similar architecture to the MNIST problem, the comparisons between the explanations provided by the tool are the same as those provided in Section 4.2.5. LIME is easier to understand since it makes use of super-pixels, SHAP is more detailed as it marks individual pixels and also shows their magnitudes, where as Lucid provides analysis into the relationship between two layers.

4.4 IMDB Sentiment analysis

The goal of the *IMDB sentiment analysis* problem is to classify whether a movie review is positive or negative. The dataset consists of 25000 *training* samples and 25000 *tests sample* where each sample is a vector of size 80×1. Each sample is a word embedding representing which words are present and in what order.

4.4.1 Model Architecture

For this problem we train a Recurrent Neural Network(RNN), the architecture can be seen in Figure 4.16. The *Embedding Layer* converts the input words into a vector representation that the model can understand, the *Long Short-Term Memory (LSTM) layer* is a recurrent layer with feedback connections which is able to keep a history of previous words in the instance so that predictions take into account previous inputs. The output layer produces probabilities that the review is either negative or positive.

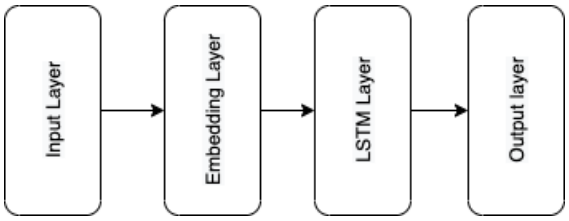


Figure 4.16: IMDB Sentiment analysis model architecture.

- Optimizer = Adam
- Learning Rate = 0.001
- Loss Function = Binary Cross-Entropy
- Training Accuracy = 99.8%
- Test Accuracy = 80.12%

4.4.2 LIME

In Figure 4.17 an attribution value is given to each individual word in the review, in this example we only show 8 words which had the largest impact. The model predicted that the review was positive with words such as “great”, “emotions” and “tears” adding towards the review being positive and by human intuition these words can have positive connotations. Negative words such as “wrong” also have obvious negative connotations, however words such as “half’ or “trying” are not

obviously negative and therefore some context is needed. Due to domain knowledge being needed, there is always some human interaction and if we have thousands of words that we are attributing on, it may become difficult to understand.

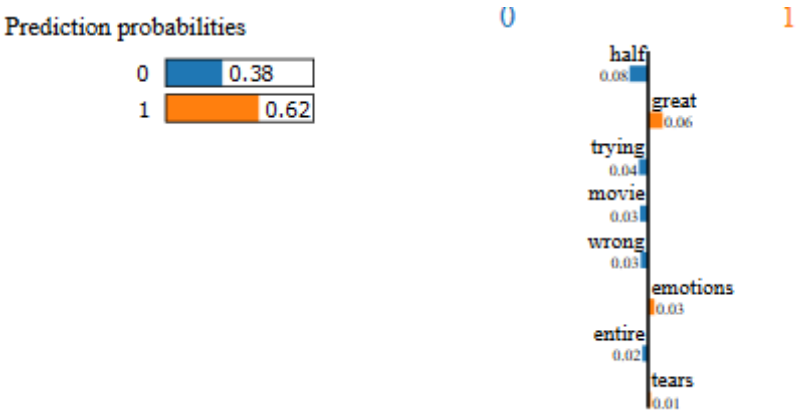


Figure 4.17: The results of the attribution where the words which had the most impact towards the prediction are sorted by magnitude. Blue represents it being a negative attribution and orange represents positive.

4.4.3 SHAP

In Figure 4.18 we have chosen *positive* to be the primary class of prediction. In this case SHAP orders the words by magnitude of attribution. The left side indicates words that attributed towards (increased the probability) of the primary class and the right side indicates words which attributed against (decreased the probability) the primary class. The output value for this particular case was 0.9 which means we have predicted that the review is positive. There are some obvious words which have positive connotation such as "'laughs". However none of the other words by human intuition are obviously good or bad if no other context is given. Therefore similar to LIME, knowledge of the domain and specific instance is needed to truly understand this explanation.

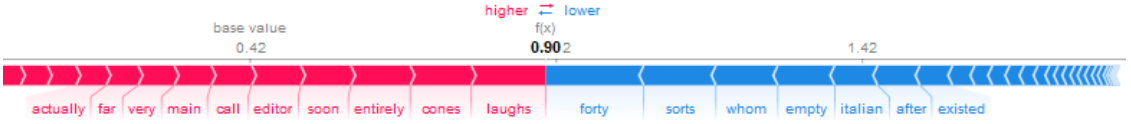


Figure 4.18: SHAP explanation for IMDB Sentiment analysis

4.4.4 Comparison

Both the explanation of LIME in Figure 4.17 and SHAP in Figure 4.18 are similar in their explanations. The main difference being again that SHAP is averaging the SHAP values over multiple background samples. The SHAP explanation is also more compact and has a lot more clarity with regards to the effect of each feature on the prediction. However both tools face the same problem in that context and domain knowledge is needed to fully understand the explanation. Explanations which contain words without obvious positive or negative connotations and reviews which are sarcastic or satirical will not be easy to understand without the proper context.

4.5 Provide intuition into LIME and SHAP

In this section we attempt to gain insight into how LIME and SHAP work by showing the relationship between a model with predefined weights and the produced feature attributions.

4.5.1 Setup

We start by creating a model with 5 weights which are explicitly defined as,

$$w_1 = 5,\ w_2 = 3,\ w_3 = 4,\ w_4 = 9,\ w_5 = 1,$$

in order to see the importance ratios between the weights we normalize them to unity,

$$w_1 = 0.227,\ w_2 = 0.136,\ w_3 = 0.18,\ w_4 = 0.409,\ w_5 = 0.045.$$

The next step is generating 500 synthetic data samples where each sample is an integer vector with 5 values between 1 and 100. We then assign each sample a probability of belonging to one of two respective classes *Positive* and *Negative* which are calculated as follows,

$$\begin{aligned}\text{Sum(Positive)} &= X[0] \times w_1 + X[1] \times w_2 + X[2] \times w_3,\\ \text{Sum(Negative)} &= X[3] \times w_4 + X[4] \times w_5,\end{aligned}$$

$$\begin{aligned}\text{Prob(Positive)} &= \frac{Sum(Positive)}{Sum(Positive) + Sum(Negative)},\\ \text{Prob(Negative)} &= \frac{Sum(Negative)}{Sum(Positive) + Sum(Negative)},\end{aligned}$$

where X is the feature vector and $X[i]$ represents the feature at index i. We pass this synthetic data to LIME and SHAP and observe how faithfully their produced feature attributions are to the actual weights.

4.5.2 LIME

Let X_1 and X_2 be 2 inputs with the following feature values,

$$\begin{aligned}X_1 &= [8,\ 24,\ 67,\ 87,\ 79],\\ X_2 &= [48,\ 10,\ 94,\ 52,\ 98],\end{aligned}$$

calculating the class probabilities for X_1,

$$\begin{aligned}\text{Prob(Positive)} &= 0.30595813,\\ \text{Prob(Negative)} &= 0.6940418,\end{aligned}$$

and for X_2,

$$\begin{aligned}\text{Prob(Positive)} &= 0.5330033,\\ \text{Prob(Negative)} &= 0.4669967,\end{aligned}$$

therefore X_1 belongs to the *Negative* class and X_2 to the *Positive* class. We pass these two inputs to the Tabular Explainer of LIME in order to extract the feature attributions from the explanations. The visualization produced by LIME shows the feature attributions rounded to 2 significant digits, however for better accuracy the comparisons will use 3 significant digits. In Figure 4.19 we can see the attribution values which LIME assigns to each class for instance X_1,

$$\phi_1 = 0.069,\ \phi_2 = 0.042,\ \phi_3 = 0.056,\ \phi_4 = 0.11,\ \phi_5 = 0.013,$$

normalizing to unity,

$$\phi_1 = 0.238,\ \phi_2 = 0.145,\ \phi_3 = 0.192,\ \phi_4 = 0.379,\ \phi_5 = 0.046.$$

The explanation for X_2 can be seen in Figure 4.20 and the attribution values are,

$$\phi_1 = 0.061,\ \phi_2 = 0.036,\ \phi_3 = 0.048,\ \phi_4 = 0.126,\ \phi_5 = 0.014,$$

normalizing to unity,

$$\phi_1 = 0.215,\ \phi_2 = 0.127,\ \phi_3 = 0.167,\ \phi_4 = 0.441,\ \phi_5 = 0.05.$$

A limitation of LIME that we have discussed before is that it only explains a single

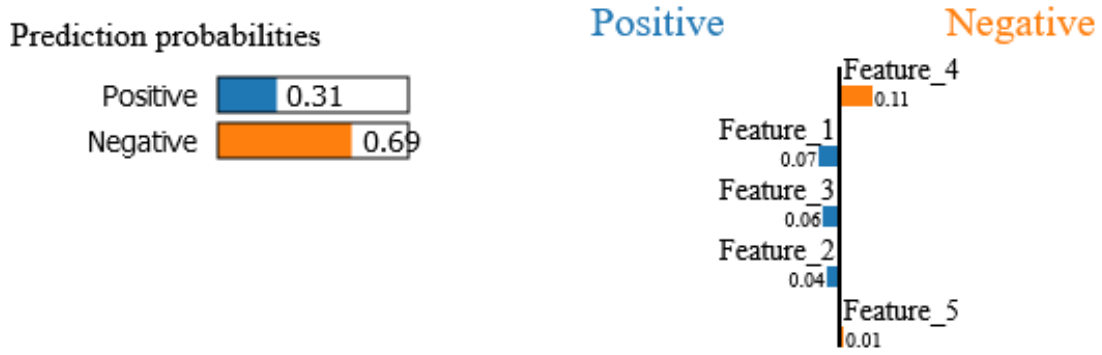


Figure 4.19: LIME explanation for input X_1

prediction at a time and we can see from the differences between the attributions

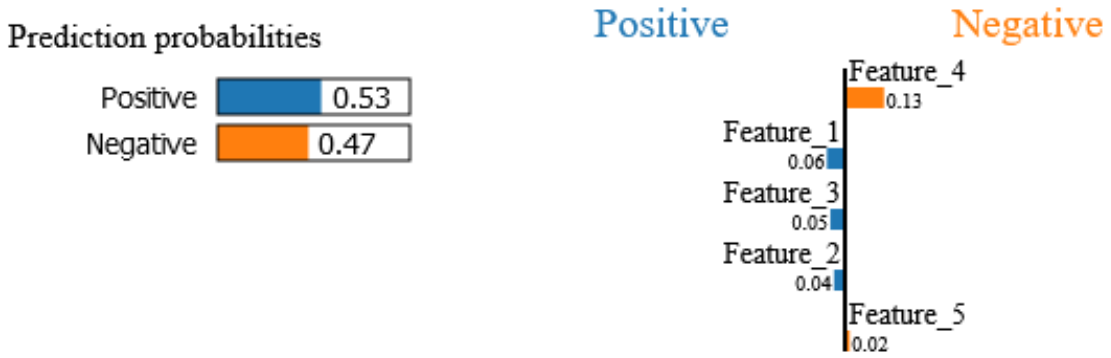


Figure 4.20: LIME explanation for input X_2

values of X_1 and X_2 that there is variance between instances. This issue could be circumvented by explaining multiple instances and averaging over the results.

4.5.3 SHAP

SHAP allows us to use the entire training set of 500 samples as background samples and gain attribution values which are averaged over the entire data set. We use *Kernel SHAP* for this experiment since our model is not a Neural Network. The feature attribution values that we have derived from SHAP and normalized to unity are,

$$\phi_1 = 0.211,\ \phi_2 = 0.126,\ \phi_3 = 0.167,\ \phi_4 = 0.44,\ \phi_5 = 0.057.$$

Figure 4.21 shows how each feature contributed towards the models prediction over the entire dataset for the *Positive* class. The more red a point is the higher that particular feature value was, where as more blue indicates a lower value. The x-axis indicates the SHAP value, where the higher the value is the more that particular feature attributed towards the probability of the *Positive* class and the lower it is the more it attributed against it. From this figure we can see that Feature 1 to 3 attributed towards the *Positive* class with Feature 1 attributing the most and Feature 2 the least. We can also see that Feature 4 attributed the most against it and Feature 5 the least. These results are inline with how we have defined the weights and from this graph we are able to visualize the different attributions

between samples and their variances.

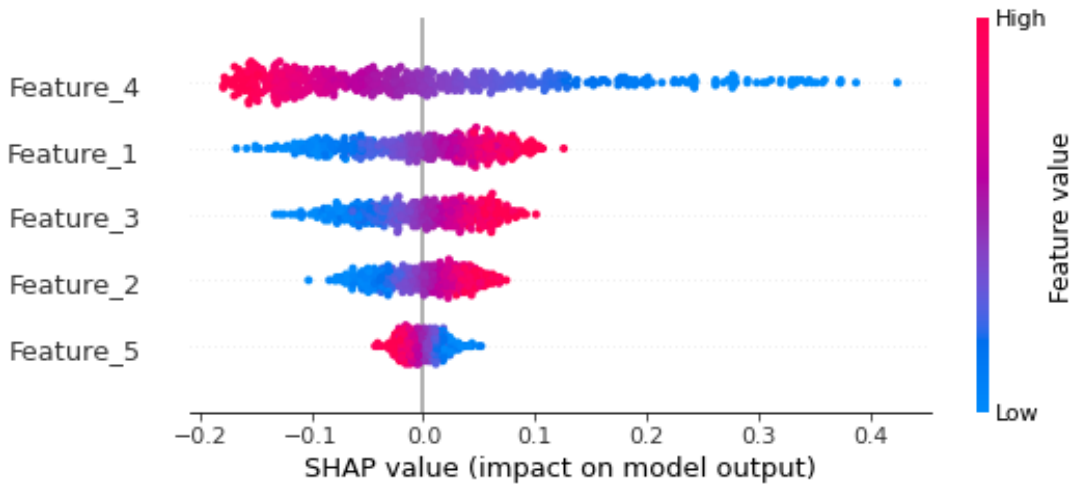


Figure 4.21: SHAP explanation for our chosen input X_1

4.5.4 Comparison

We have tabulated the results for comparison purposes in Figure 4.22. We have also graphed the weights and attributions in Figure 4.23 for better visualization. We have also taken the average LIME attributions over the entire dataset in order to compare it to SHAP. We can see that the attribution values are fairly accurate in describing the defined weights and the ratios of feature importance are comparable to those of our model. From the results we have gathered, we can conclude that LIME and SHAP are able to generate fairly accurate linear estimations of a linear model by only using it's prediction results. These results allow us to gain some trust in LIME and SHAP, however further experimentation is needed to see if complex neural networks can also be approximated well.

4.6 Conclusion

From our evaluation we have shown that in all instances SHAP is superior to LIME in terms of the value that the explanations offer. In this thesis we are interested in

Type	Weight_1	Weight_2	Weight_3	Weight_4	Weight_5
Actual	0.227	0.136	0.182	0.409	0.045
LIME_X_1	0.238	0.145	0.192	0.379	0.046
LIME_X_2	0.215	0.127	0.167	0.441	0.05
LIME_Average	0.205	0.121	0.163	0.456	0.055
SHAP	0.211	0.126	0.167	0.44	0.057

Figure 4.22: Comparison of weights extracted from the different tools.

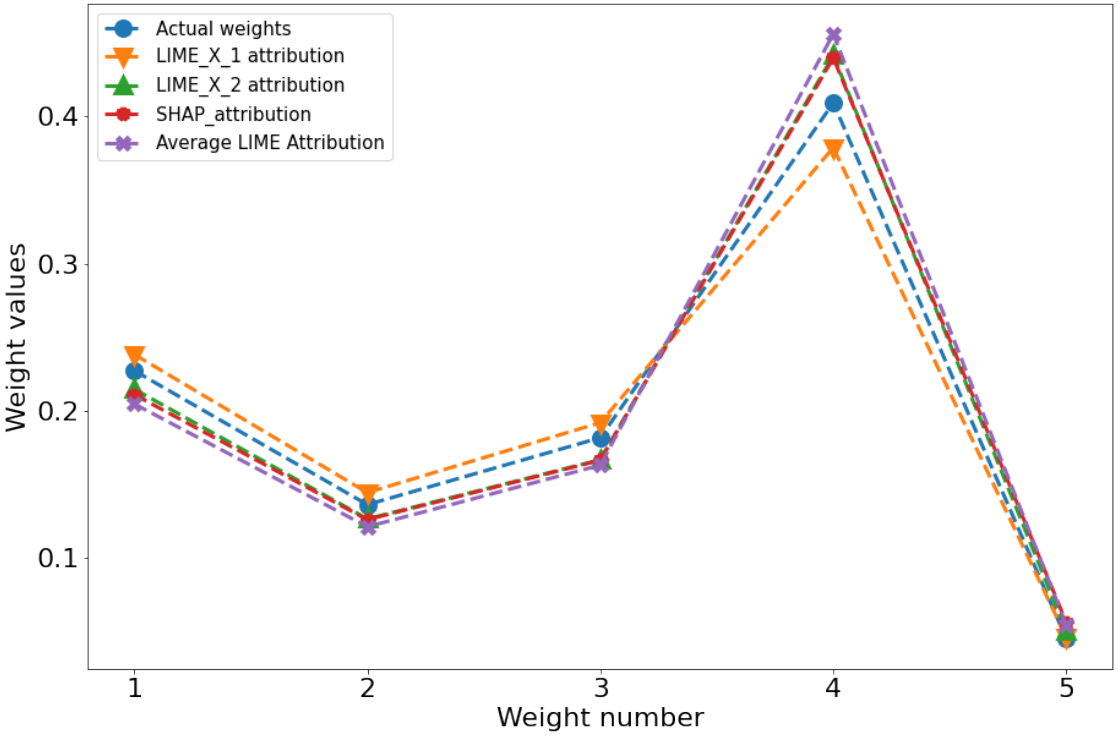


Figure 4.23: Graph of weights extracted from the different tools.

how faithful these explanations are and therefore the extra computation of SHAP is not a deciding factor. One of the major advantages that SHAP has is that it is able to provide SHAP values that are averaged over multiple instances. This can somewhat be achieved with SP-LIME however it only chooses individual instances that are distinct from one another and the values are still calculated for each individual instance. Therefore we will solely be using SHAP to explain our Credit Loan Neural Network in Chapter 5.

Chapter 5

Credit Case Study

5.1 Problem Statement

Companies which operate in the credit environment want to see returns on their investments. Before a loan is granted to a prospective client the appropriate risk analysis needs to be performed. In many instances this is a manual process where a credit employee has to assess a clients credit portfolio and ascertain if they are eligible for a loan and if so, on what terms. As most manual processes it can be slow and prone to human-error. By adopting machine-learning we are able to eliminate the human risk factor and significantly increase the speed. Linear modelling techniques such as *Logistic Regression* are the preferred choice due to them being inherently interpretable. Interpretability is a strict requirement due to the need to understand why the model is making the decisions that it does. Linear models are however very limited in what they can do and by using Neural Networks or other nonlinear models we may be able to significantly increase the accuracy at identifying risk, however such techniques are generally seen as *black-boxes*. In this chapter we use SHAP in order to gain some interpretability into credit risk Neural Networks so that they could be considered as possible solutions.

5.2 The Data set

Up until this point we have used our tools on *toy* examples where the data has been engineered to work well with machine-learning. In this chapter we will be using a proprietary credit risk dataset which has been provided to us by *Praelexis* which aims to simulate actual features used by credit companies to calculate potential client risk. Our dataset consists of a total of 200000 client samples where 28768 defaulted on their loan and the remaining 171232 did not. As we can see we have a heavy class imbalance in the data with a ratio of roughly 85 : 15. In practice this ratio of imbalance can be a lot higher [54]. The trained model may be skewed to more likely predict the most present class. How we solve this issue is discussed in Section 5.6.4.

5.3 Aim

Our aim is to see if we can provide enough interpretability into a credit Neural Network so that it may be considered as a possible alternative to linear models by credit companies looking to adopt machine-learning. In order to achieve this we train a *Logistic Regression* model which is inherently interpretable and a generic feedfoward Neural Network and compare their interpretability. The goal of these models is to predict a probability that a client will default on a loan given their input features. As we have seen in Chapter 4, SHAP is superior to LIME in every way but speed and therefore we shall use SHAP as our preferred tool for interpretation.

5.4 Weights of evidence

The weights-of-evidence (WOE) transformation [55] is a nonlinear transformation of the original variables. It is used to measure the strength of each attribute at separating good and bad loans. It can be seen as the measure of the difference in the proportioning of good and bad within each attribute (i.e., the odds of a person of that attribute being good or bad) [55]. Continuous variables are first converted to categorical variables through a process called *binning*. Using the information from the training data we calculate the WOE value for each bin. Each client is

either considered to be good (0) or bad (1). The WOE can be calculated as,

$$\text{WOE} = \left[\ln\left(\frac{\text{Distr Good}}{\text{Distr Bad}}\right)\right] \tag{5.1}$$

The steps of calculating WOE can be therefore be summarized as follows:

1. If the variable is continuous separate the data according to the distribution into multiple parts known as *bins*.
2. Calculate the number of good and bad loans in each bin.
3. Calculate the % of good and bad loans in each bin.
4. Calculate the WOE of each bin by using (5.1).
5. Replace the raw variable values in the input with the WOE value of the bin that it belongs to.

We can see an example of creating WOE bins for a variable in Table 5.1. The *boundaries* are the bin ranges and can either be manually or automatically chosen. The *Distr good* and *Distr bad* is extracted from the training dataset and by using (5.1) we are able calculate the WOE value for each bin. For example suppose that our input variable has a value of 1.5. Therefore it falls into the $(1, 2]$ bin and the value 1.5 will be replaced with the WOE value -0.07. In Figure 5.1 we can see an example of WOE binning for a binary classifier. The axis are the values of the two features. The red star and blue dot represent the 2 classes. The shaded parts are the values which will be assigned to the 2 classes after the WOE transformations. This illustrates how the WOE transformation allows us to construct a nonlinear decision boundary.

5.4.1 Benefits of WOE

- Missing values are separated into their own bins and is therefore easier to handle.
- Provides a linear relationship with log odds.

- It can treat outliers. Extreme values which fall outside the average range are given their own bin.
- We are able to determine the stability of our model by comparing the WOE values of the training and unseen data.
- Converts continuous variables into categorical variables so that there is no distinction between them.

For our dataset we have chosen to use *automatic binning* because we want the preprocessing steps to be as automated as possible. This is to prevent our domain knowledge from affecting the produced explanations.

	Bin	Boundaries	Count good	Distr good	Count bad	Distr bad	WoE
	1	$(-\infty, 1]$	1760	0.0973	798	0.2033	−0.37
	2	$(1, 2]$	5238	0.2896	1223	0.3116	−0.07
	3	$(2, 3]$	7881	0.4357	1034	0.2634	0.50
	4	$(3, \infty)$	3210	0.1775	870	0.2217	−0.22
Total			18089	1.0	3925	1.0	

Table 5.1: WOE example

5.5 Metrics

Before we discuss our models architecture we introduce the metrics with which we use to measure the performance of the model. Firstly we introduce some terminology used within the metrics definitions. To provide some context we will be using our problem of predicting whether a client will default on their loan or not. We refer to clients which will default on their loan as *defaulting clients* and those which will not default as *regular clients*. Defaulting clients that have been correctly identified by the model are referred to as *True Positives (TP)*, where as regular clients which are incorrectly identified as defaulting clients are referred to as *False Positives (FP)*. Regular Clients which are correctly identified are referred to as *True Negatives (TN)*, where as defaulting clients which are incorrectly identified as regular clients are referred to as *False Negatives (FN)*.

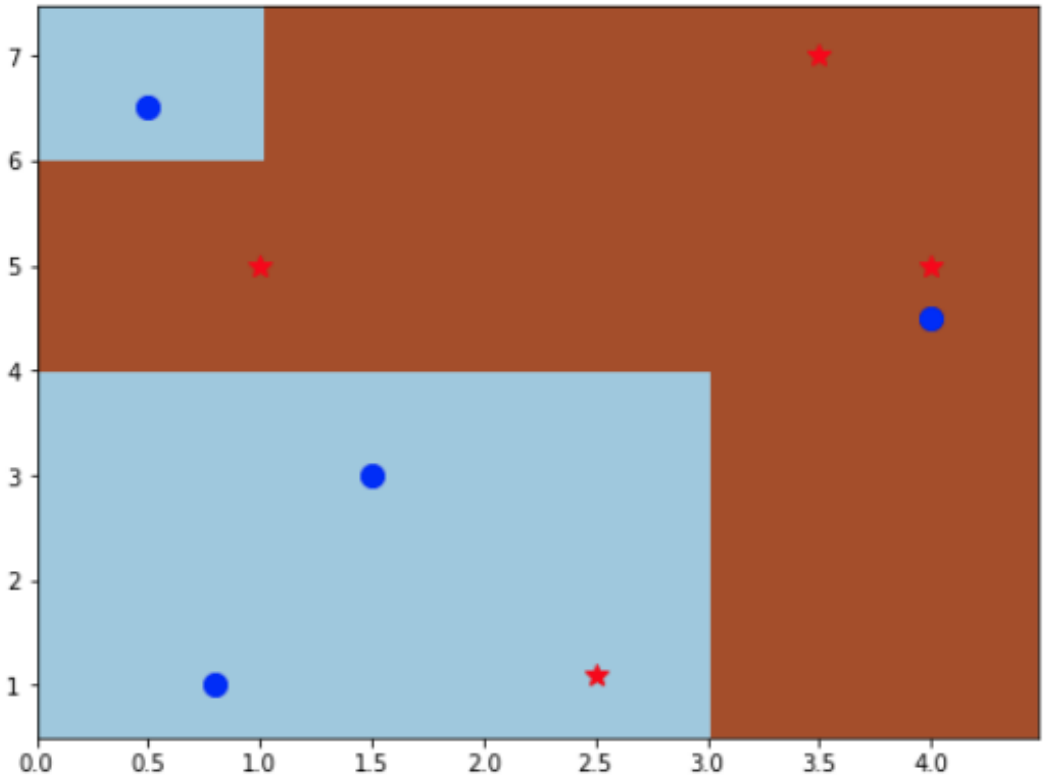


Figure 5.1: The decision boundary constructed by weights of evidence binning.

5.5.1 Accuracy

In most examples the common metric that is used to measure how well a model performs is the *accuracy* measure, however it does not work as well if the dataset has a class imbalance. Suppose in the previous example of identifying fraudulent claims that the training dataset consisted of 95% legitimate claims and 5% fraudulent claims. If we engineer the model to simply predict every claim to be legitimate then we will achieve a 95% accuracy, however the model would never be able to identify fraudulent claims and therefore does not solve our problem. Although a high accuracy does not necessarily mean a model is performing well, a low accuracy can be indicative of a poor performing model and therefore has relevance. Using the previously defined terms we can define accuracy as,

$$\text{Accuracy} = \frac{TN + TP}{FN + FP + TN + TP} \tag{5.2}$$

5.5.2 Precision

An important metric is called *precision* which is the proportion of positive identifications that were actually correct. In the previous example of identifying fraudulent claims, precision would be the percentage of *actual* fraudsters in the group of people that the model has predicted to be fraudsters. Therefore using this measure we are able to determine the ratio of how many of our regular clients we have falsely predicted to be a fraudster. We define precision as,

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.3}$$

5.5.3 Recall

Recall is similar to precision in that it is the proportion of actual positives that were identified correctly. In the example, recall would be the percentage of fraudsters the model was able to find in the entire *pool* of fraudsters. Therefore we define recall as,

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.4}$$

5.5.4 F1 Score

The *F1 Score* is the harmonic mean between the precision and the recall. It is usually used in conjunction with accuracy as a single measure of a models performance. If we are looking to create a model balanced between recall and precision then the F1 score is the metric to use. The F1 Score ranges from 0 to 1. 1 indicates perfect precision and recall where as 0 is when either the precision or recall is 0.

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}} \tag{5.5}$$

5.5.5 Trade offs

Although precision and recall both seem like metrics that we should aim to maximize in our model, in practice it is usually not possible. We have to select the metric which is most valuable to us based on the problem and use that as the main performance measure while keeping the others in mind. In the case of our model

which detects defaulting clients we have to weigh whether marking regular clients as defaulting clients (Precision) or identifying less actual defaulting clients overall (Recall) is more detrimental to our business. A possible solution is assigning a cost to each error made in our predictions in order make more informed decisions.

5.6 Preprocessing

5.6.1 Feature Standardization

Before we begin training our model we first have to *standardize* the features. This entails subtracting the mean and scaling to unit variance so that the standardized data has a mean(μ) of 0 and a standard deviation(σ) of 1,

$$X_{standardized} = \frac{X - \mu}{\sigma} \tag{5.6}$$

This is needed so that features have a variance within the same magnitude, because if one feature has a variance magnitudes larger than others it might dominate the objective function and in turn make the estimator rely too much on it and unable to learn from other features.

Effect on feature attributions

Standardizing the values changes their meaning to how far they are from the mean in terms of standard deviations. This means it is possible for strictly positive features to take on negative values when standardized. We have to account for this when observing the explanations.

Weights of evidence

For the WOE models the distributions of good and bad are used, which is already a form of standardization. Therefore we only perform explicit feature standardization for the raw input models.

5.6.2 Splitting the dataset

Before we start using the data we first have to split the dataset into 3 isolated sets namely *training set*, *validation set*, and *test* set. The test and validation set will each encompass 20% of the total data and the training set the remaining 60%. The training set is what we use for training the model, the validation set is used to fine-tune any parameters specific to the model such as the regularizer coefficient which we will discuss in Section 5.6.3, and the test set is only used at the end in order to evaluate the models performance on unseen data. Since we have a heavy class imbalance in the dataset, we use a *stratified* split which allows each of the 3 sets to maintain the ratio of class imbalance present in the overall dataset.

5.6.3 Feature Selection

Since there are 33 features present in the dataset, we look to reduce the number of features by carefully removing those which are better explained by other features or add nothing of value to the model. For the model which uses the raw features we will make use of a simple linear classifier with an L1 penalizer for feature selection. Since by default L1 regularization is able set weight coefficients to 0 we are to able remove weights which have little or no contribution to the model. For the WOE model, we will be using Information Values extracted from the WOE for feature selection.

Benefits of feature selection

- Decreased training time.
- More stability if correlated variables are removed.
- Improve the classification scheme by removing low contributing features.
- Explanations provided by SHAP are easier interpreted due to less features needing to be explained.

Information value

The Information Value (IV) [55] of a variable can be considered as the total predictive strength of the variable and is an indication of that variable's ability to separate between good and bad loans. It can be seen as the symmetric alternative to the *Kullback-Leibler (KL) divergence* [56]. The IV can be calculated using the following formula,

$$\text{IV} = \sum_{i=1}^{n} (\text{Distr Good}_i - \text{Distr Bad}_i) \cdot \text{WOE}_i \tag{5.7}$$

substituting the equation for WOE from (5.1),

$$\text{IV} = \sum_{i=1}^{n} (\text{Distr Good}_i - \text{Distr Bad}_i) \cdot \ln\left(\frac{\text{Distr Good}}{\text{Distr Bad}}\right)_i \tag{5.8}$$

where the sum is taken over all bins n. A good indication of how the IV of a variable relates to their predictive strength is shown in Table 5.2. By calculating

IV range	Interpretation
< 0.02	Not predictive
$[0.02, 0.1)$	Weak predictive
$[0.1, 0.3)$	Medium predictive
$[0.3, 0.5)$	Strong predictive
> 0.5	Suspicious

Table 5.2: Interpretation of IV values.

the IV of each of the features we are able to keep features within a certain threshold and remove the rest. For our case we have chosen the minimum threshold as 0.02 and maximum threshold as 0.6. Looking at Figure 5.2 we can see the IV of all the features and which features have been chosen to be excluded and included according to our thresholds. We have decreased the number of features from 33 to 22.

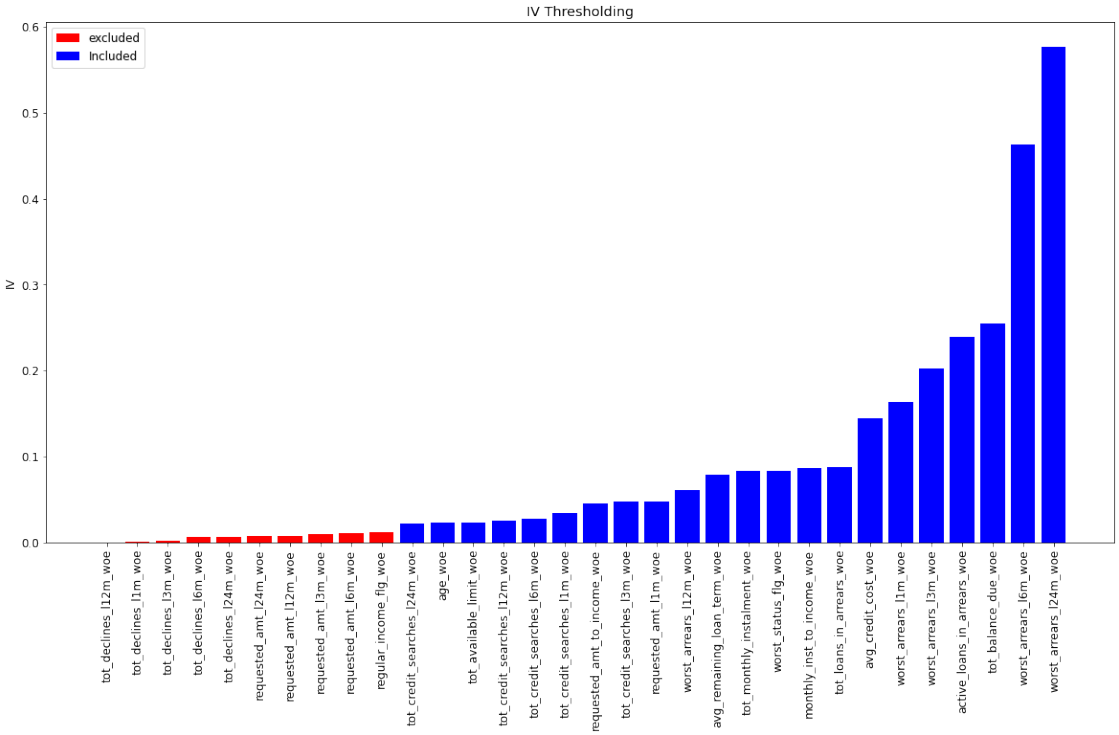


Figure 5.2: Information Value Thresholding on input features.

L1 Regularizer

For the raw features we use a model trained with a L1 regularizer which is referred to as a *Lasso* model for feature selection. L1 regularization is able to assign higher weighting to features which play a larger role in classification and in turn can set those that do the least to 0. Given a linear model,

$$Y = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_0, \tag{5.9}$$

where Y is the label, x_i are the features and β_i are the weight coefficients. The cost function can be written as,

$$\sum_{i=1}^{m} (Y_i - \sum_{j=1}^{n} \beta_j x_{ij})^2, \tag{5.10}$$

where m is all the input samples in the training set and n is the number of features in a sample. We add a regularization term with the L1 penalizer,

$$\sum_{i=1}^{m}(Y_i - \sum_{j=1}^{n}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{n}|\beta_j|, \tag{5.11}$$

where λ is a coefficient that we can tune to adjust how aggressively we constrain the weights. Figure 5.3 demonstrates the constraint region built by the L1 regularizer for a model which contains two weight coefficients β_1 and β_2. $\hat{\beta}$ is the unconstrained least squares estimate. The red ellipses are the contours of the least squares error function. The blue area represents the feasible region $|\beta_1| + |\beta_2| \leq t$ of the constraints introduced by the penalty where t is the coefficient of the regularizer. The possible values are those where the contour and diamond meet and as we can see at 4 points of the diamond one of the weight coefficients are 0. This extends to higher dimensions where there are more possibilities of weights being 0. We are looking for the intersection of the red ellipses and the blue region as the objective is to minimize the error function while maintaining the feasibility. Let p be the number of features and therefore the dimensionality, in this case we have $p = 2$. When $p > 2$ the diamond becomes a rhomboid and has many corners flat edges and faces which have the opportunity to become 0.

Selecting the regularization coefficient

As discussed previously the higher we set the value of the regularization term the stricter we constrain the weight coefficients. Therefore the larger λ is the stricter the feature selection is as more coefficients will be set to 0. In order to select a λ that provides us with the best features we need to tune this as a hyper parameter by using our validation set. We need to make sure that we do not throw away any features that have information that cannot be explained by the selected features. Therefore we tune λ by making it stricter and monitoring how it affects the performance of the model. In Figure 5.4 we have plotted the results of the *accuracy*, *recall*, and *precision* metrics against a stronger penalty. Note that the L1 coefficient in this case is inversely proportional to how strict the regularizer is. As we can see both the recall and precision take quite a significant drop once λ reaches

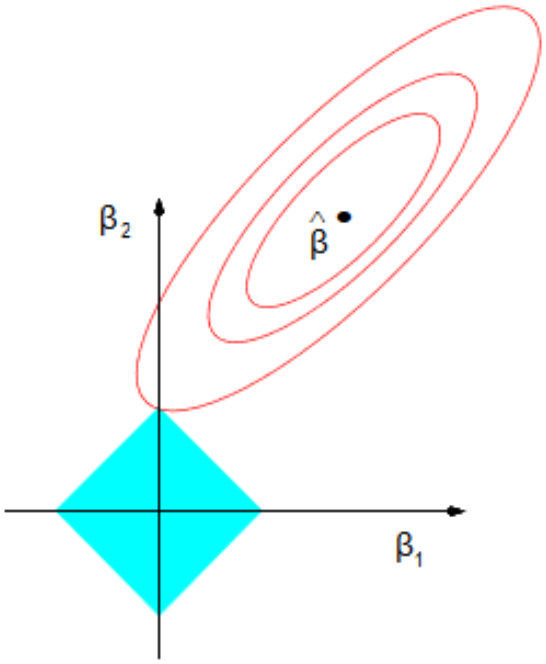


Figure 5.3: Boundary of lasso model. **Image from** [57].

0.0001. Therefore 0.001 is the smallest that we can set λ before we start noticing a drop in the performance of the model. Setting λ to 0.001 reduces the number of features from 33 to 17. This is a big reduction while hardly losing any strength within the model, this may not be the most stable approach if we are looking to productionize this model but it allows the interpretation to be simpler to showcase due to the reduced number of attributions. These features are described in Section 5.9.

5.6.4 Class imbalance

Class imbalance has a significant effect on conventional classification techniques because they assume a balance of classes [58]. In credit companies it is expected that there is a noticeable class imbalance within their data because they can not afford to have many bad loans [59][54]. This results in there being far more loans which could be considered *good*. This makes it difficult for many machine-learning techniques to learn the boundary between a good and bad loan because it does

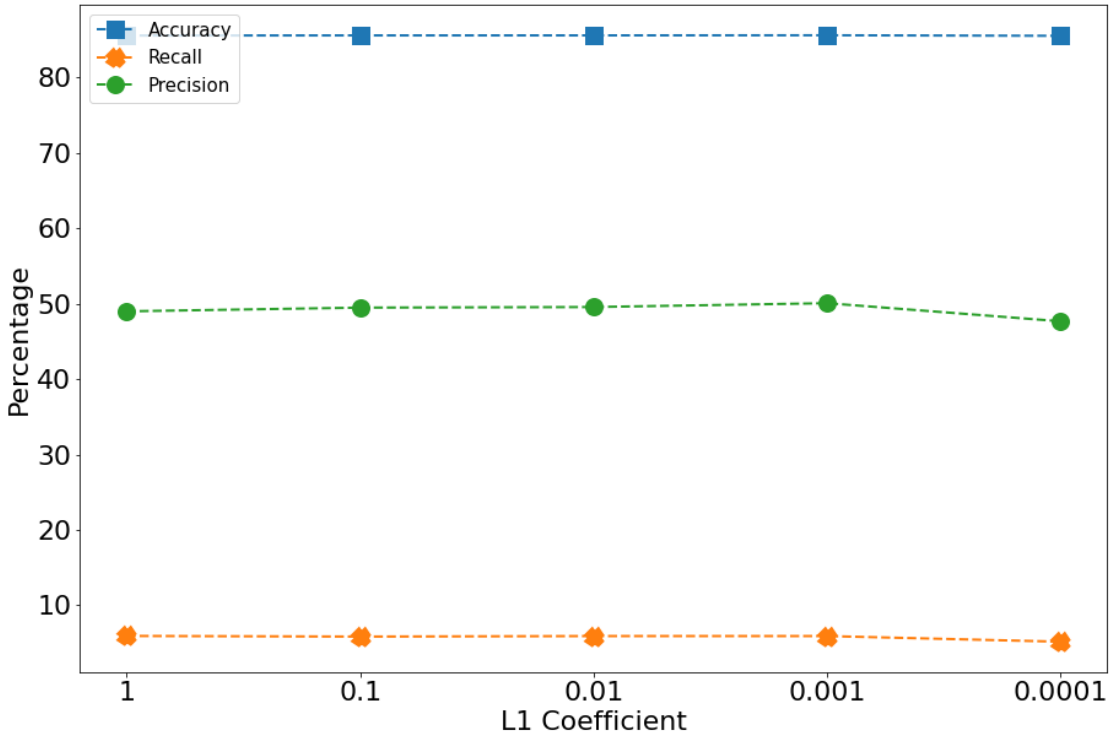


Figure 5.4: Choosing the L1 coefficient.

not have enough bad loans to properly identify them. Selectively sampling from the data in order to create a balanced dataset [45] is a possible solution however this results in a far smaller training set and thus has an overall negative impact if we have a small sample size. It can be shown that by providing misclassification costs to our model provides an increase in the performance of classifiers [60]. If we suppose that misclassifying a client that belongs to the *default* as the *not default* class is r times as serious as the reverse. We can provide a weighting which penalizes such misclassifications r times as heavily as the reverse. In order to minimise the overall weighted misclassification rate we have to minimise,

$$\text{Assign to class 1 if } p(1|x) > (1+r)^{-1} \text{and to class 0 otherwise} \tag{5.12}$$

We can incorporate these misclassification costs into our model by providing a weighting of classes which penalizes the model more for predicting the less present class incorrectly which is the *default* class in our case. By using these bias class

weights we may be able to identify more clients which may default, however we lose some accuracy in predicting non-defaulting clients.

$$\text{Class weights} = \frac{|X|}{|y| \cdot [y_1, ..., y_n]} \tag{5.13}$$

Where $|X|$ is the number of samples, $|y|$ is the number of classes, y_i is the number of labels of class i. By using the values in our dataset we get,

$$\begin{aligned} \text{Class weights} &= \frac{120000}{2 \cdot [102739, 17261]} \\ &= [0.584, 3.476] \end{aligned} \tag{5.14}$$

by using these class weights the model is penalized roughly 5.95 times more for incorrectly classifying *defaulting* clients as *not defaulted*. This causes the weights to be optimized to predict the default class more. For the raw input models we will use versions with and without these biased class weights and compare their results and how their interpretations differ.

5.7 Models

5.7.1 Logistic Regression

Logistic Regression is a widely used technique. Even some Deep Neural Networks use it within their output layer. It is known to work well for binary classification problems and provides a soft prediction.

Soft Predictions

A soft prediction refers to a prediction which gives the probability that a input belongs to a specific class, where as a hard prediction simply assigns a 0 or 1 regardless if the prediction was on the boundary between the two classes. In Figure 5.5 we can see the difference between the thresholding mechanisms used in hard and soft predictions, the x-axis represents the output before we threshold and the y-axis is the value after it goes through the activation function. When using a hard threshold the values are either set to 0 or 1. For soft threshold the value is set as

a probability between 0 and 1, this allows us to see the confidence of the model in the prediction. For example in Figure 5.5 if the value is 0, hard thresholding will force it to 1, however soft thresholding would set it to 0.5 which indicates that it could belong to either class.

Making a prediction using Logistic Regression

Given a general linear model,

$$y = \sum_{i=1}^{n} w_i \cdot x_i + w_0, \tag{5.15}$$

where n is the number of features in the input. We introduce the logistic function, or more commonly referred to as *sigmoid* [61],

$$\text{Sigmoid}(z) = \frac{1}{1 + e^{-z}} \tag{5.16}$$

Substituting in (5.15), this can be rewritten as,

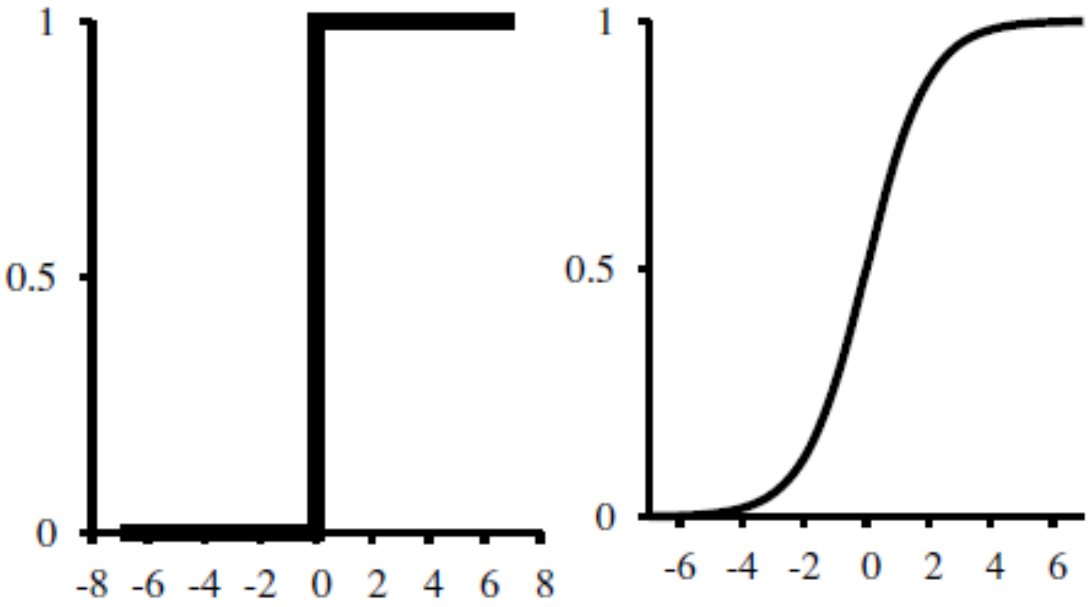
$$\text{Sigmoid}(w \cdot x + w_0) = \frac{1}{1 + e^{-(w \cdot x + w_0)}} \tag{5.17}$$

Once the model is trained, we substitute the input vector into (5.17) in order to make a prediction.

Cross entropy cost function

The cost function can be seen as a measurement of how incorrect the model is at estimating the relationship between the input X and their corresponding labels y. It can also be seen as the distance between the predicted labels and the actual labels. Let the sigmoid function of the model be $h_w(x)$ and the cost function be $J(w)$ then,

$$J(w) = \begin{cases} -\log(h_w(x)) & \text{if } y = 1 \\ -\log(1 - h_w(x)) & \text{if } y = 0 \end{cases}$$



(a) Hard Threshold

(b) Soft Threshold

Figure 5.5: Comparison of hard and soft thresholds. **Image from**[61].

can be condensed as,

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log(h_w(x^{(i)})) + (1-y^{(i)})log(1-h_w(x^{(i)}))\right] \tag{5.18}$$

Now that we have the cost function, we need to somehow decrease it until the weights converge.

Gradient Descent

In order to find the optimal parameters for the weights we have to minimize the cost function using gradient descent. Let y be the true labels and $h_w(x)$ be the predicted labels. When using gradient descent, we set the weights to some initial value either randomly or by some initialization algorithm. We can update the weights by subtracting the derivative of the cost function,

$$w_j \longleftarrow w_j - \alpha \cdot \nabla_w J(w) \tag{5.19}$$

where α is referred to as the *learning rate* which is how large the steps we take when updating the weights are. Since h_w is the sigmoid function we can simplify this to,

$$w_j \longleftarrow w_j - \alpha \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{5.20}$$

This continues until the weights have converged. In Figure 5.6 we can see how the weights update during gradient descent. We start at some initial weights and after each weight update iteration we try to reach the minimum of the cost function. When we reach the minimum, the weights are considered to have converged and the training is completed as we have found the optimal parameters. Note that most if not all of the popular optimization procedures in Neural Networks is based on this simple idea, because the gradient can be efficiently calculated using *back-propagation*. It should also be noted that numerous important modifications have been made for the sake of efficiency and robustness.

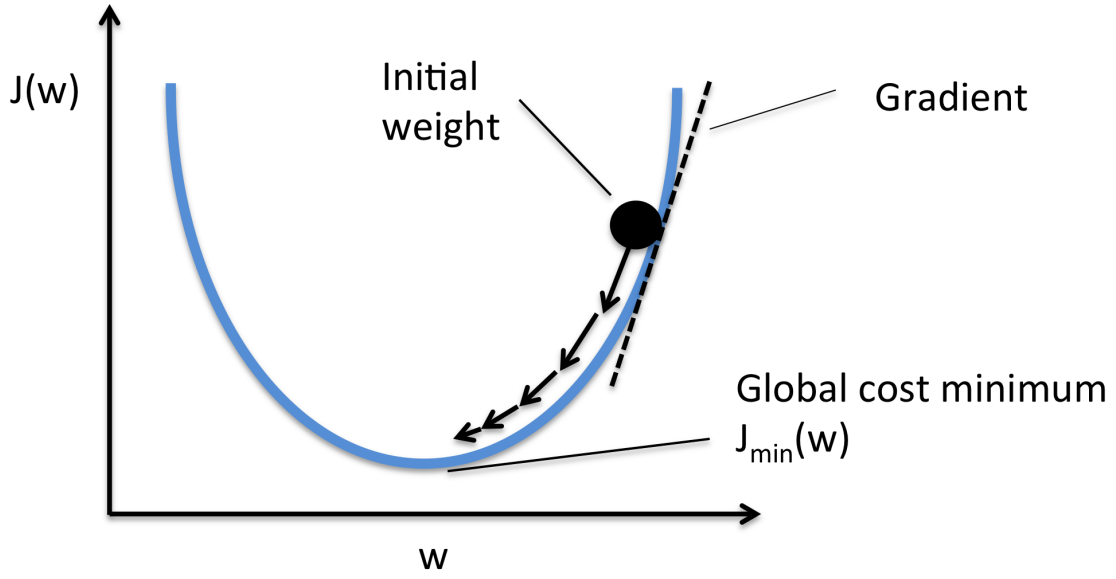


Figure 5.6: How Gradient Descent is performed. **Image from** [5]

Interpreting Logistic Regression

Since the outcome of Logistic Regression is a probability between 0 and 1, we can not simply look at the weight coefficients as a direct interpretation. The weighted sum in the linear equation (5.15) is transformed into a probability by the sigmoid function (5.17). Therefore the weights do not affect the probability linearly. Introducing the log odds function,

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right) = \log\left(\frac{P(y=1)}{P(y=0)}\right) = w_0 + w_1x_1 + ... + w_nx_n \tag{5.21}$$

We can adjust this equation in order to determine how the prediction changes when one feature x_j is changed by 1 unit [62]. Applying the exp function to both sides we end up with,

$$\frac{P(y=1)}{1-P(y=1)} = odds = \exp(w_0 + w_1x_1 + ... + w_nx_n) \tag{5.22}$$

Comparing the ratio of two predictions when increasing a single feature by 1,

$$\frac{odds_{x_j+1}}{odds} = \frac{\exp^(w_0 + w_1x_1 + ... + w_j(x_j+1) + ...w_nx_n)}{exp^(w_0 + w_1x_1 + ... + w_jx_j + ...w_nx_n)} \tag{5.23}$$

Then applying the following exponential rule,

$$\frac{\exp(a)}{\exp(b)} = \exp(a-b) \tag{5.24}$$

and removing many terms,

$$\frac{odds_{x_j+1}}{odds} = \exp(w_j(x_j+1) - w_jx_j) = \exp(w_j) \tag{5.25}$$

The *Odds Ratio (OR)* can be seen as a measurement of the strength between 2 events. Given 2 events A and B, we can define the OR as the ratio of the odds of A occurring in the presence of B compared to the odds of A occurring in the absence of B [63].

- OR=1 The events A and B are independent.
- OR> 1 If B is present then A has higher odds of occurring .

- OR<1 If B is absent then A has higher odds of occurring.

Therefore according to (5.25) a change in "a feature by one unit changes the odds ratio (multiplicative) by a factor of" [62] $\exp(w_j)$. Another interpretation is that by changing a feature's value "by one unit increases the log odds ratio by the value of the corresponding weight" [62]. Interpretation is different depending on the feature type [62]:

- *Numerical features*: If you increase the value of feature x_j by one unit, the estimated odds change by a factor of $\exp(w_j)$ [62].
- *Binary categorical features*: We refer to one of the categories as the *reference* category. Changing the feature x_j from the reference category to the other category in turn changes the estimated odds by a factor of $\exp(w_j))$ [62].
- *Categorical feature with more than two categories*: When dealing with multiple categories one-hot-encoding is commonly used. For a feature which has N categories we would need N-1 columns in our one-hot-encoder. The N-th category is considered the reference category. The interpretation for each category then is equivalent to the interpretation of binary features [62].

In Table 5.5 we can see an example of a Logistic Regression classifier trained to predict the probability of cervical cancer given certain risk factors [62]. Looking at the *Num. of diagnosed STDs* feature which is numerical, increasing the number of STDs by 1 would in turn increase the odds of cancer vs. no Cancer by a factor of 2.26. Looking at the feature *Hormonal contraceptives y/n* which is a binary categorical feature indicates that for women who use hormonal contraceptives, the odds for cancer vs. no cancer are by a factor of 0.89 lower, compared to women without hormonal contraceptive. It is important to note that these interpretations are only true if every other feature stays the same.

5.7.2 Feedforward Neural Network

We will be training a generic feedforward Neural Network with a single hidden layer. The architecture can be seen in Figure 5.7. Every layer in the Neural Network is *Fully Connected* which means that every incoming node is connected

	Weight	Odds ratio	Std. Error
Intercept	−2.91	0.05	0.32
Hormonal contraceptives y/n	−0.12	0.89	0.30
Smokes y/n	0.26	1.29	0.37
Num. of pregnancies	0.04	1.04	0.10
Num. of diagnosed STDs	0.82	2.26	0.33
Intrauterine device y/n	0.62	1.85	0.40

Table 5.3: "The results of fitting a logistic regression model on the cervical cancer dataset. Shown are the features used in the model, their estimated weights and corresponding odds ratios, and the standard errors of the estimated weights" [62]. **Table from** [62].

to every outgoing node. The input layer is simply the input features. The *hidden layer* contains 9 nodes for the raw features and 11 nodes for the WOE network. For both networks the output layer is a single node that uses the *sigmoid* activation function which is the probability that the client will default. The Neural Network only has a single hidden layer and is considered very small when compared to modern networks.

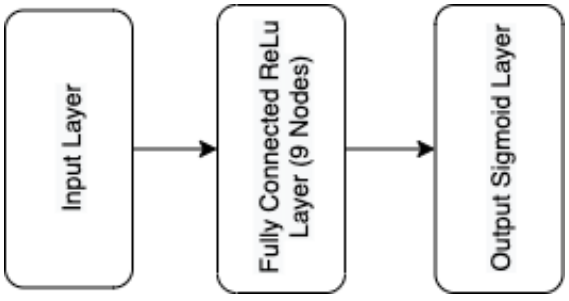


Figure 5.7: Architecture of credit Neural Network.

5.7.3 Trainable weights

In order to show that a Neural Network is theoretically more powerful than Logistic Regression, we have to compare their *trainable weights*. Trainable weights are the weights which models will estimate in order to make a prediction. Theoretically the more weights we can estimate, the more accurate the predictions will be. However

it is important that we account for *overfitting* which is when the model corresponds too closely to the training data and thus fails to predict unseen data reliably. In practice there are other factors such as the networks architecture and preprocessing techniques which have an effect. We can calculate the trainable weights with,

$$\text{LR} = \text{Input weights} + \text{Intercept}$$
$$\text{NN} = \text{Input weights} \cdot \text{Hidden weights} + \text{Hidden weights} \cdot \text{Output weights}$$

We can see the comparison of the trainable weights between the models in Figure 5.4.

Model	**Input**	**Hidden**	**Output**	**Intercept**	**Trainable Weights**
Logistic Regression	17	0	1	✓	18
Logistic Regression WOE	22	0	1	✓	23
Neural Network	17	9	1	×	162
Neural Network WOE	22	11	1	×	253

Table 5.4: Comparison of Trainable Weights between the different model architectures

5.8 Results

In this Chapter we will compare the following models:

- Logistic Regression with raw inputs.
- Logistic regression with raw inputs and altered class weights.
- Logistic Regression with WOE inputs.
- Neural Network with raw inputs.
- Neural Network with raw inputs and altered class weights.
- Neural Network with WOE inputs.

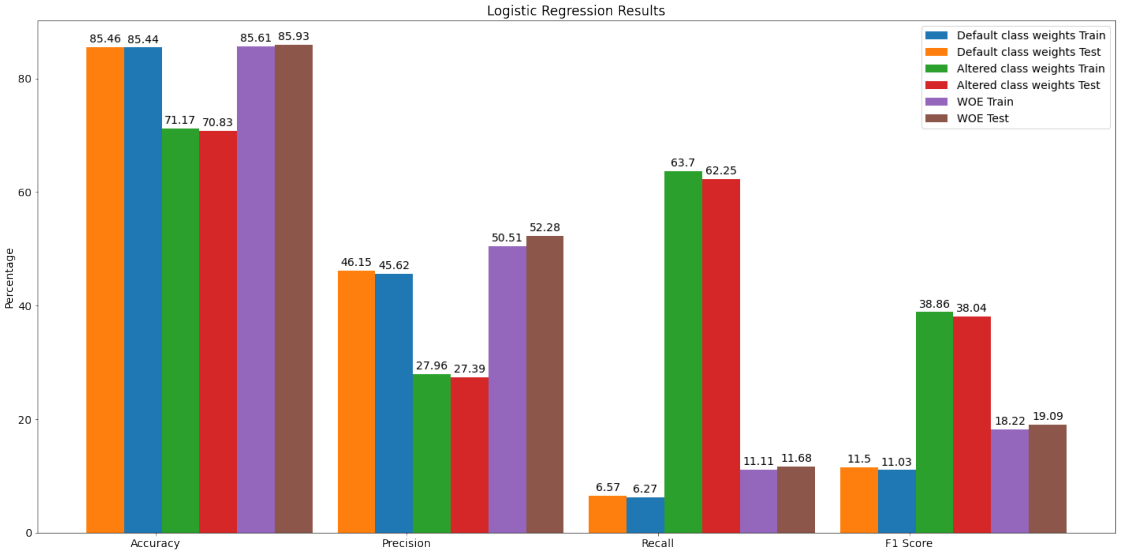


Figure 5.8: Results of the Logistic Regression classifier.

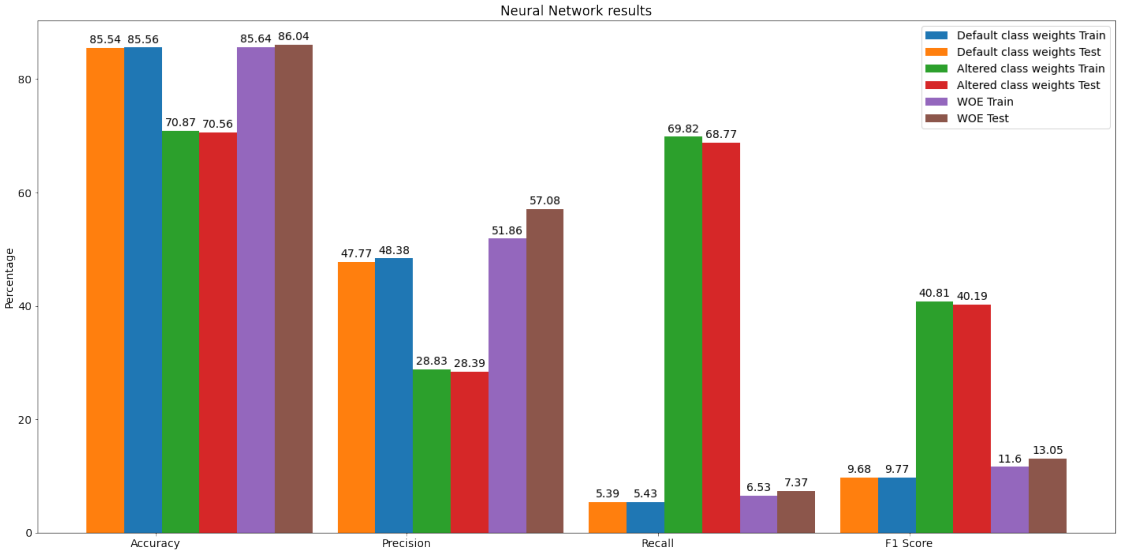


Figure 5.9: Results of the Neural Network.

5.8.1 Comparison

Comparing the results from the Logistic Regression classifier in Figure 5.8 and Neural Network classifier in Figure 5.9 their *accuracy* is mostly the same with the only real difference being a slight increase in *precision* and *recall* in the Neural Network which also in turn results in a difference in *F1 scores*. From this it may seem that there is no reason to use a Neural Network classifier, however there are plenty of optimizations and various other network compositions that may provide better results. Our purpose is to provide interpretability and therefore we are not concerned with building the most robust network possible.

5.8.2 Normal vs Bias class weights

In both models when using the bias class weights discussed in Section 5.6.4 we see on average a 15% drop in accuracy which is quite significant, however an enormous increase in *recall* which means the bias versions are better at identifying clients which *defaulted*. For credit companies where the purpose is to identify clients which may default on their loans, the bias weights provide better results. In Section 5.10 we compare both the normal and bias class weights models and observe the explanations produced by SHAP in order to compare the difference in feature attributions.

5.8.3 Raw Inputs vs WOE

When looking at the comparisons from the raw input features to the WOE transformed features in both the Logistic Regression results in Figure 5.8 and the Neural Network results in Figure 5.9 there does not some to be a significant difference. The only noticeable difference is a slight increase in precision. WOE transformation are the standard when working in the credit industry and from these results it may seem like there isn't much merit in it, but we will see in the *explanations* in Section 5.10 that there is a significant difference.

5.9 Feature descriptions

In order to make sense of the feature attributions we first have to describe what each feature means. From our credit domain knowledge we are able to specify which features magnitudes increase proportionally to the risk that the client will default.

5.9.1 Not Default

The following features decrease the risk of the client defaulting,

age (For clients who are middle-aged)
: The age of the client.

regular_income_flg
: A flag which indicates whether the client has a regular income or not.

tot_available_limit
: Total monthly limit available to client.

5.9.2 Default

The following features increase the risk of the client defaulting,

age (For clients who are very young or old)
: The age of the client.

requested_amt_l1m
: Requested amount over a month.

requested_amt_l24m
: Requested amount over 24 months.

requested_amt_to_income
: Ratio of requested amount to monthly income.

avg_credit_cost
: Average cost of credit.

avg_remaining_loan_term

The average amount of months left on the clients loans.

tot_monthly_instalment

Monthly installment paid towards the loan.

monthly_inst_to_income

Ratio of income to monthly installment.

tot_credit_searches_l1m

Total credit searches in the last month.

tot_credit_searches_l3m

Total credit searches in the last 3 months.

tot_credit_searches_l6m

Total credit searches in the last 6 months.

tot_credit_searches_l12m

Total credit searches in the last 12 months.

tot_credit_searches_l24m

Total credit searches in the last 24 months.

worst_status_flg

Flag of whether this loan is the most in arrears.

tot_declines_l12m

Total credit declines in the last 12 months.

active_loans_in_arrears

Active loans that are in arrears.

tot_loans_in_arrears

Total loans that are in arrears.

worst_arrears_l1m

Worst arrears in the past month.

worst_arrears_l3m
Worst arrears in the past 3 months.

worst_arrears_l6m
Worst arrears in the past 6 months.

worst_arrears_l12m
Worst arrears in the past 12 months.

worst_arrears_l24m
Worst arrears in the past 24 months.

tot_balance_due
Total balance due on the loan.

5.10 Interpretation

In this section we observe how the inherent interpretability of Logistic Regression compares to the explanations provided by SHAP for the Neural Network. We provide interpretation for both the normal and bias class weights models as well as a comparison between the attributions generated by the WOE features compared to the raw features.

5.10.1 Logistic Regression

As we have discussed in Section 5.7.1 for Logistic Regression the weight coefficients are not enough to provide interpretability. We have to extract the weight coefficients w from the linear equation (5.15) and also calculate their odds ratio (which is described in Section 5.7.1) with (5.25). We have tabulated the weight coefficients as well as their respective odds ratios. We have also graphed the odds ratio for extra clarity. Table 5.5 and Figure 5.10 is for the raw input model. Table 5.6 and Figure 5.11 is for the raw input model with altered class weights. Lastly Table 5.7 and Figure 5.12 is for the WOE model.

WOE weight coefficients

From Table 5.7 we can see that all of the weight coefficients are positive for the WOE classifier. It may seem like every variable positively contributes towards the *default* class. This is however not the case as we do not know the range of the magnitudes of the variables by just looking at this table. If the woe value of a feature in a specific bin is negative then even if the weight coefficient is positive it will provide a negative attribution. Therefore Table 5.7 can not be considered sufficient information as we would need to view the range of the feature values to discern whether a feature decreases or increases the probability of *defaulting*.

Feature	**Weight**	**Odds ratio**
requested_amt_l24m	0.0028	1.002804
tot_declines_l12m	0.0098	1.009848
requested_amt_to_income	0.0109	1.010960
tot_credit_searches_l12m	−0.0207	0.979513
worst_arrears_l24m	0.0233	1.023574
tot_available_limit	−0.0305	0.969960
tot_credit_searches_l3m	0.0335	1.034067
age	−0.0474	0.953706
tot_loans_in_arrears	0.0565	1.058127
regular_income_flg	−0.0645	0.937536
tot_monthly_installment	−0.0659	0.936224
worst_status_flg	0.0676	1.069937
avg_credit_cost	0.0781	1.081231
monthly_inst_to_income	0.0832	1.086759
avg_remaining_loan_term	0.0846	1.088282
worst_arrears_l6m	0.1027	1.108159
active_loans_in_arrears	−0.2181	0.804045

Table 5.5: Weight coefficients for Logistic Regression.

5.10.2 Neural Network

For the Neural Network we will be using *Deep SHAP* as it leverages the composition of Neural Networks to significantly increase the speed of SHAP. We will be looking at how SHAP explains individual predictions as well as the explanation for the entire model.

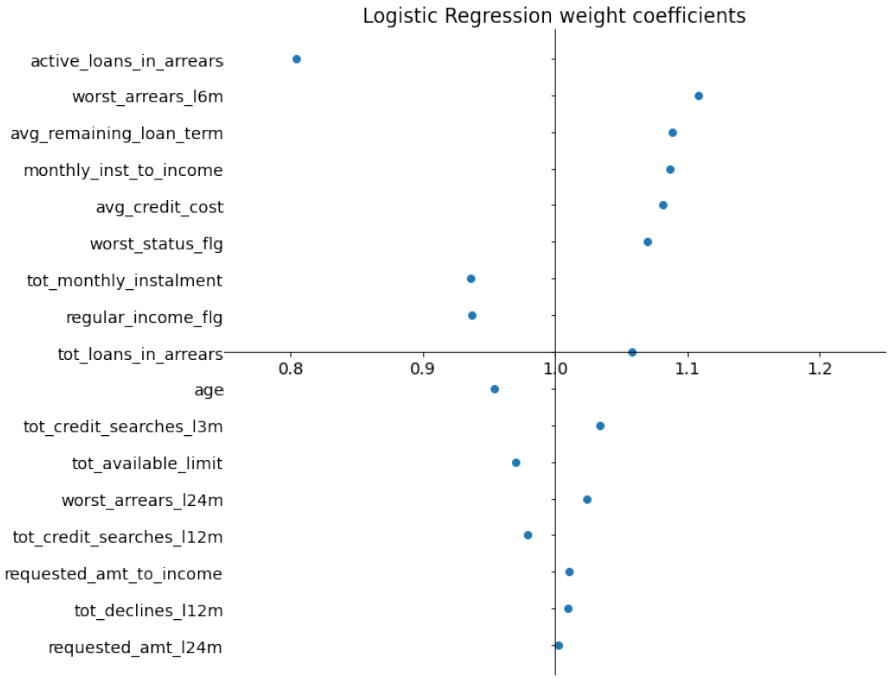


Figure 5.10: Odds ratios for Logistic Regression.

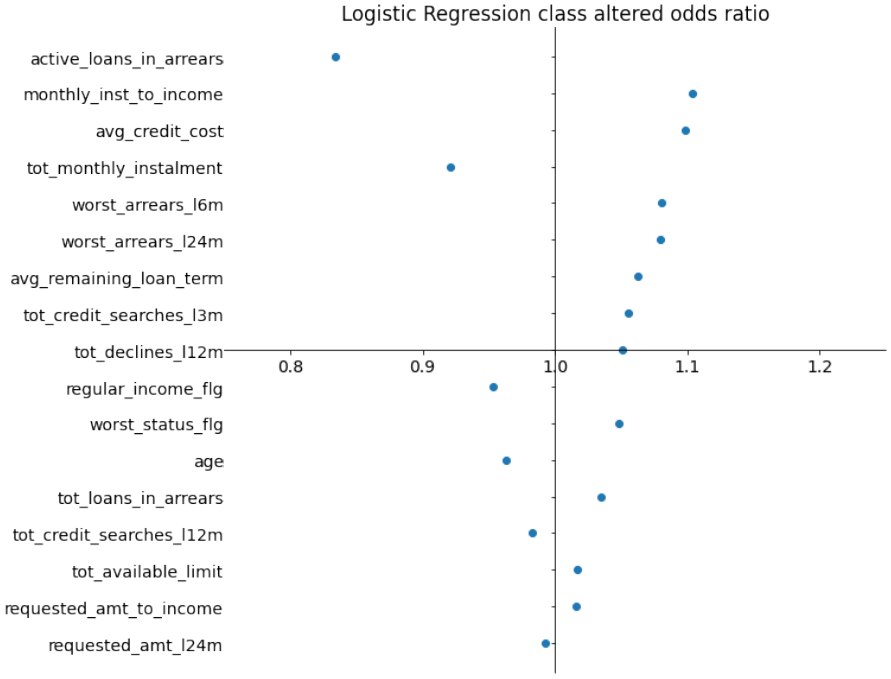


Figure 5.11: Odds ratios for Logistic Regression with altered class weights.

Feature	Weight	Odds ratio
requested_amt_l24m	−0.0076	0.992429
tot_declines_l12m	0.0497	1.050956
requested_amt_to_income	0.0161	1.016230
tot_credit_searches_l12m	−0.0172	0.982947
worst_arrears_l24m	0.0769	1.079934
tot_available_limit	0.0171	1.017247
tot_credit_searches_l3m	0.0539	1.055379
age	−0.0375	0.963194
tot_loans_in_arrears	0.0339	1.034481
regular_income_flg	−0.0481	0.953038
tot_monthly_installment	−0.0819	0.921364
worst_status_flg	0.0475	1.048646
avg_credit_cost	0.0941	1.098670
monthly_inst_to_income	0.0989	1.103956
avg_remaining_loan_term	0.0609	1.062793
worst_arrears_l6m	0.0773	1.080366
active_loans_in_arrears	−0.1814	0.834102

Table 5.6: Weight coefficients for Logistic Regression with altered class weights.

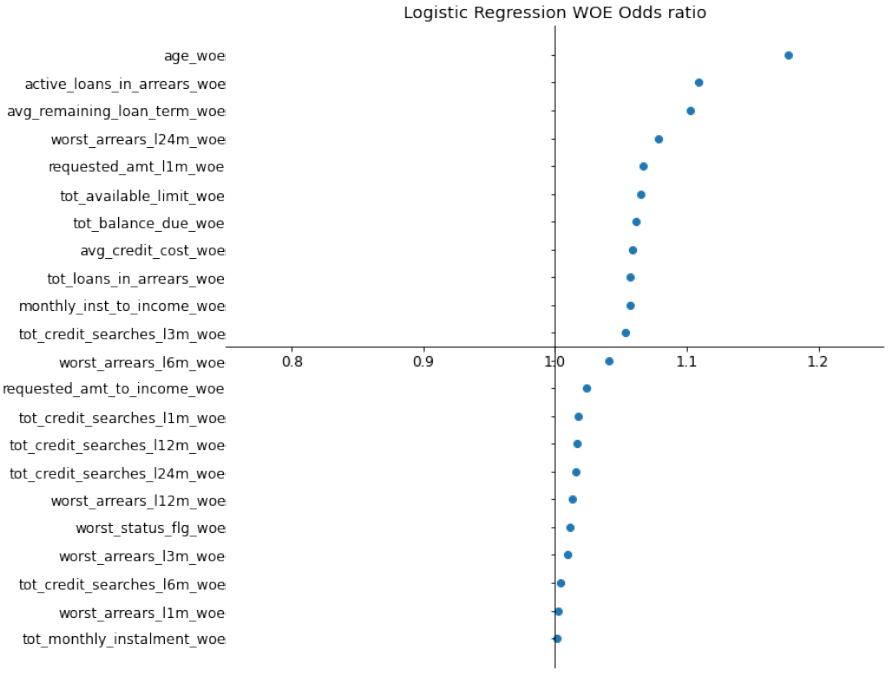


Figure 5.12: Odds ratios for WOE Logistic Regression.

Feature	Weight	Odds ratio
tot_monthly_instalment_woe	1.001401	0.0014
worst_arrears_l1m_woe	1.002102	0.0021
tot_credit_searches_l6m_woe	1.004108	0.0041
worst_arrears_l3m_woe	1.009041	0.0090
worst_status_flg_woe	1.010859	0.0108
worst_arrears_l12m_woe	1.012984	0.0129
tot_credit_searches_l24m_woe	1.015621	0.0155
tot_credit_searches_l12m_woe	1.016332	0.0162
tot_credit_searches_l1m_woe	1.017145	0.0170
requested_amt_to_income_woe	1.024188	0.0239
worst_arrears_l6m_woe	1.040603	0.0398
tot_credit_searches_l3m_woe	1.053376	0.0520
monthly_inst_to_income_woe	1.056541	0.0550
tot_loans_in_arrears_woe	1.057280	0.0557
avg_credit_cost_woe	1.059079	0.0574
tot_balance_due_woe	1.061412	0.0596
tot_available_limit_woe	1.064814	0.0628
requested_amt_l1m_woe	1.067159	0.0650
worst_arrears_l24m_woe	1.078100	0.0752
avg_remaining_loan_term_woe	1.102963	0.0980
active_loans_in_arrears_woe	1.109268	0.1037
age_woe	1.176919	0.1629

Table 5.7: Weight coefficients for WOE Logistic Regression.

Model explanation

We start by explaining on a global level the attributions the model assigns to each input feature and compare it to the Logistic Regression explanations. Figure 5.13 is the explanation for the normal network, Figure 5.14 is for the bias class weights network, and Figure 5.15 is the WOE network. Rather than just giving a single value for attributions SHAP is able to plot over multiple instances in the dataset and showcase how each feature attributes to different instances. The y-axes are the feature names and the x-axes are their corresponding SHAP values, larger values impact the model more. Positive values attribute towards the *default* class and negative values to the *not default* class. The hue of a point ranges from blue to red where the redder a point the larger the magnitude of that feature was in that particular instance.

Explanation Using the information from Section 5.9 we know that the larger the magnitude of the *Age* feature the more it contributes against the client defaulting. In Figure 5.13 and 5.14 we can see that the larger the magnitude of age the SHAP value falls more into the negative region, this indicates that the older the client is, the less likely they are to *default*. This coincides with our knowledge. On the inverse we know that the *avg_credit_cost* feature contributes towards the client *defaulting*, this is also shown in Figure 5.13 and 5.14 where the larger the magnitude of *avg_credit_cost* is, the larger it's SHAP value is and thus the higher contribution it had towards predicting that the client would default.

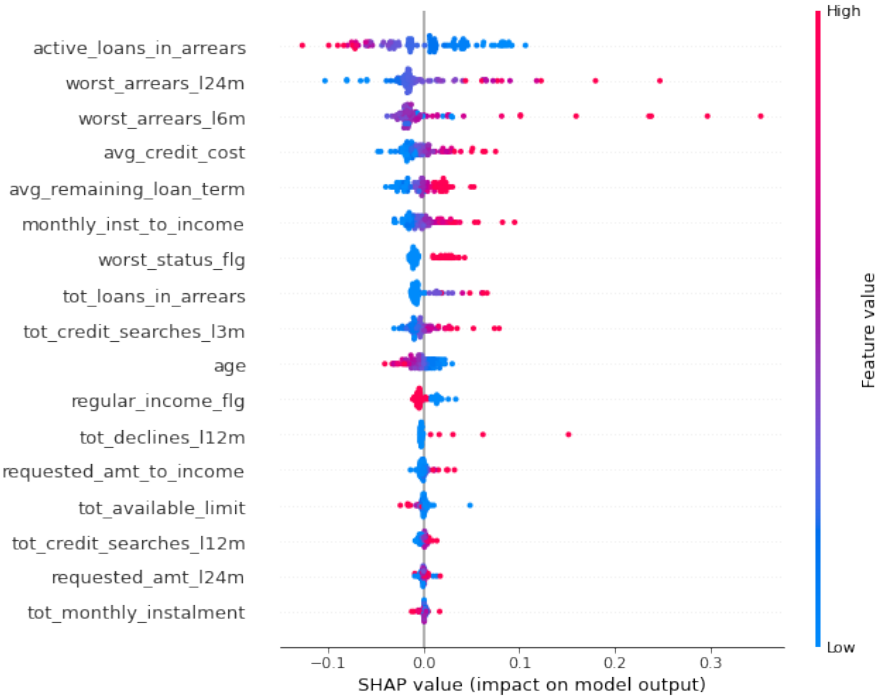


Figure 5.13: Deep SHAP interpretation for Neural Network.

Individual predictions

An advantage of using a tool such as SHAP is that we are able to obtain explanations for individual predictions rather than just for the entire model. This is useful for credit employees as this allows them to obtain explanations for an individual

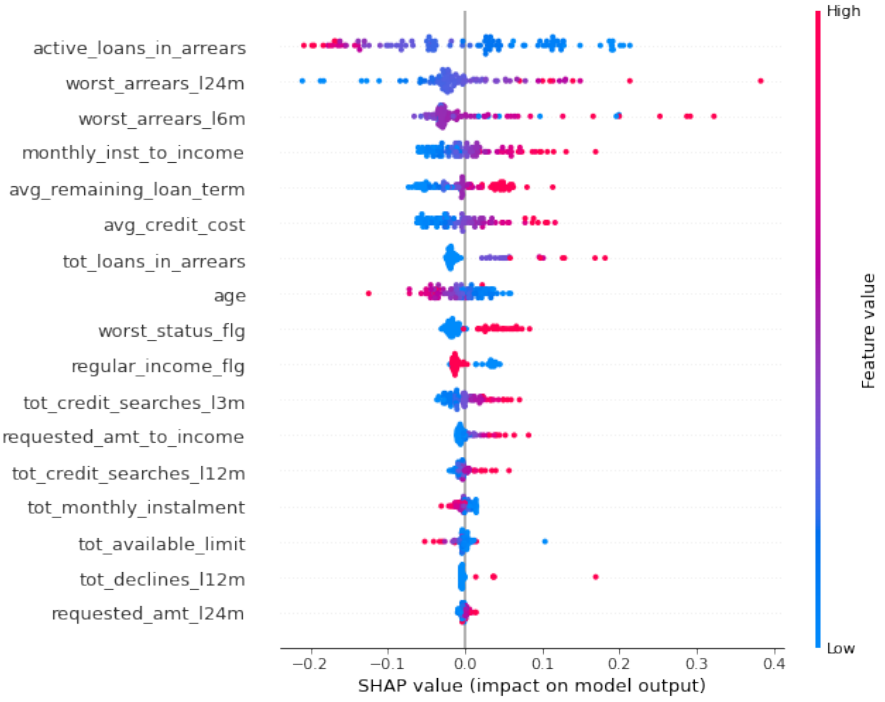


Figure 5.14: Deep SHAP interpretation for class weight bias Neural Network.

client. We have chosen a single client which has *defaulted* and in Figure 5.16, 5.17 and 5.18 we can see a plot of the explanation for it's prediction. Figure 5.16 is the *normal* class weights and the output of the model is that it is 27% certain that the client will *default on their loan*. Figure 5.17 is the explanation for the *bias* class weights model and it is 71% certain that the client will *default*. Figure 5.18 is the explanation for the WOE Neural Network and it is 37% that the client will *default*. The y-axes are the input features and the x-axes is the model output value. The bottom of the line is the average model output value, which is the models default output value given that there is no information about the input features. As the line reaches the top, each feature either adds or subtracts from the models predicted value, therefore features that subtract from the value are negative attributions and those that add are positive attributions. The numbers in brackets are the values of that feature of this particular client standardized.

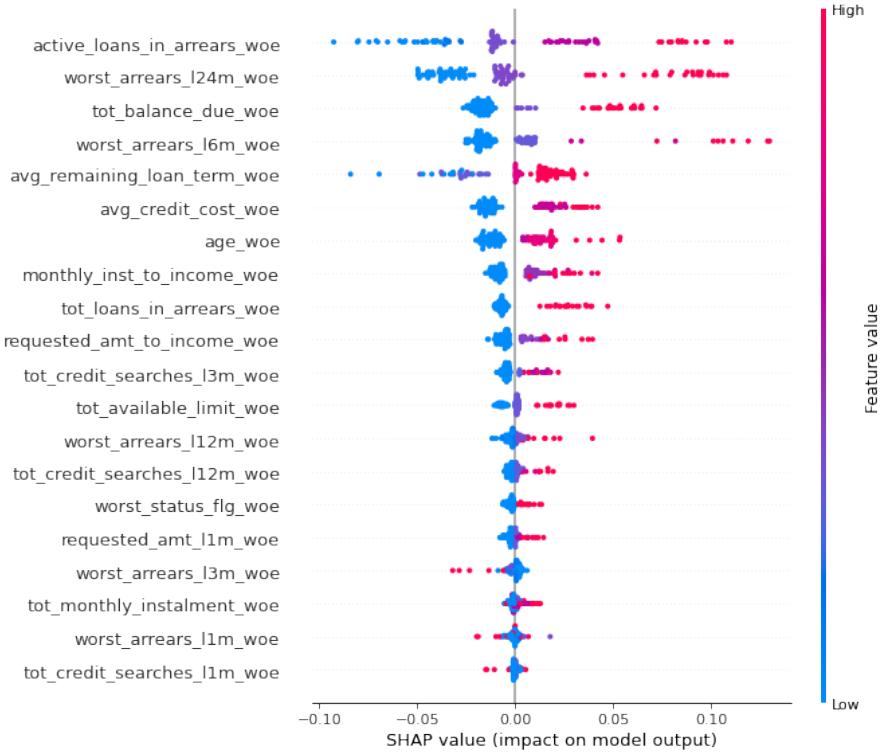


Figure 5.15: Deep SHAP interpretation for WOE Neural Network.

5.10.3 Comparison

Comparing the weight coefficients obtained from Logistic Regression with the SHAP explanations we can see that although the SHAP values have a different meaning than the raw weight coefficients they are comparable. They both provide a relative value of how much that particular feature would affect the prediction of the model. The SHAP explanations are far more descriptive as they are able to provide explanations over multiple instances, which shows consistency and we are also able to provide explanations for individual instances. Since the SHAP values are consistent with what we expect we can conclude that we have successfully provided explanations into this previously considered *black-box* credit risk Neural Network. It is important to note that there is no known way to quantitatively measure the difference between the Logistic regression weight coefficients and the SHAP values produced from the Neural Network. If further research is to be done it is important to further explore possible methods that can provide quantitative comparisons.

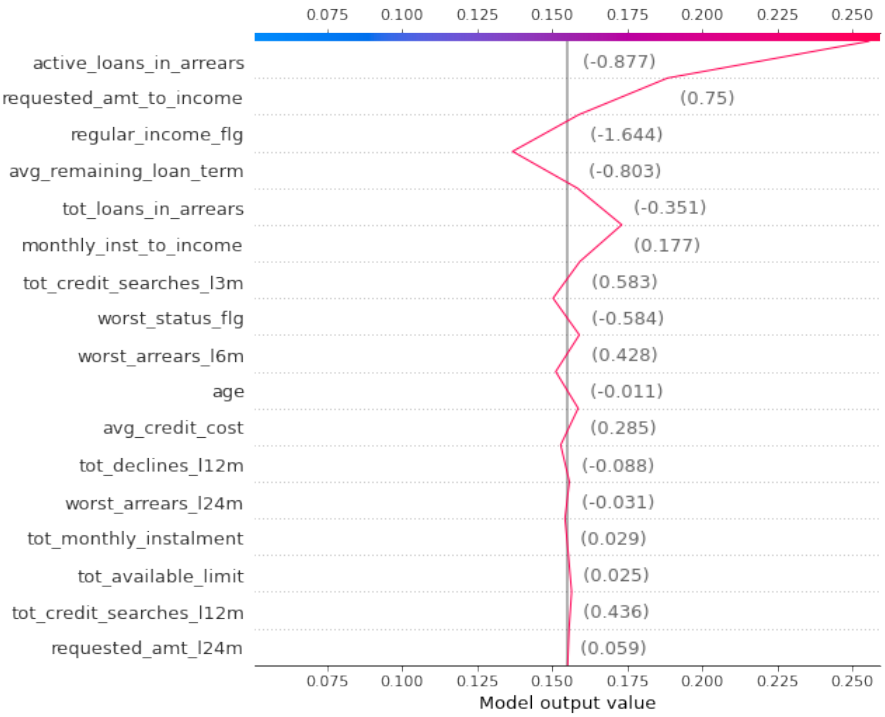


Figure 5.16: Single prediction Deep SHAP interpretation for Neural Network.

5.10.4 Exposing a problem with the raw input models

If we refer back to Section 5.9 we can see that the *active_loans_in_arrears* feature is the number of active loans that the client currently has in arrears. An increase in its feature value would in turn cause the risk of the client to increase. As we can see for the WOE model in Figure 5.15 the larger the magnitude of *active_loans_in_arrears* the higher the contribution towards the *Default* class as expected. However looking at the SHAP explanations for the raw input models in Figure 5.13 and 5.14 the larger *active_loans_in_arrears* is the less likely the client is to *default* . This means that there is an obvious flaw present with the raw input models with regards to this feature. This is not obvious from just observing the Logistic Regression models since it hard to determine how the feature reacts over multiple instances and different magnitudes by only looking at the features overall contribution. Even though the performance of the two models are relatively the same by viewing the SHAP explanations produced, we were able to identify this problem.

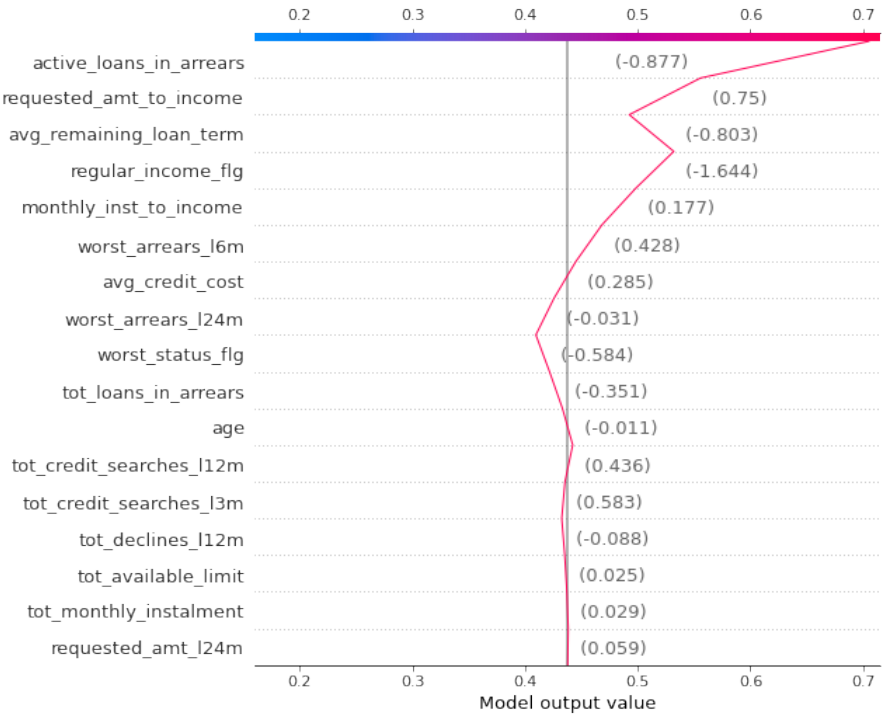


Figure 5.17: Single prediction Deep SHAP interpretation for class weight bias Neural Network.

5.10.5 Monitoring changes within the architecture

Figure 5.19 is the SHAP explanation of the WOE Neural Network with the hidden layer removed form the network itself. When comparing this to Figure 5.15 there is a noticeable difference as the *worst_arrears_l24m_woe* has the largest contribution. The explanation is expected to change but SHAP can provide some insight to what the changes are. When training a neural network one of the hardest problems is figuring out what the architecture should be. An example would be for a simple neural network how do we decide how many hidden layers? What about how many neurons in each layer? Usually we would just use our metrics and experiment with the architecture. However metrics don't really tell the whole story and with SHAP we can observe how the architecture affects the variables attributions themselves. Thus by using SHAP we are able to determine how a change in the networks architecture affects the feature attributions.

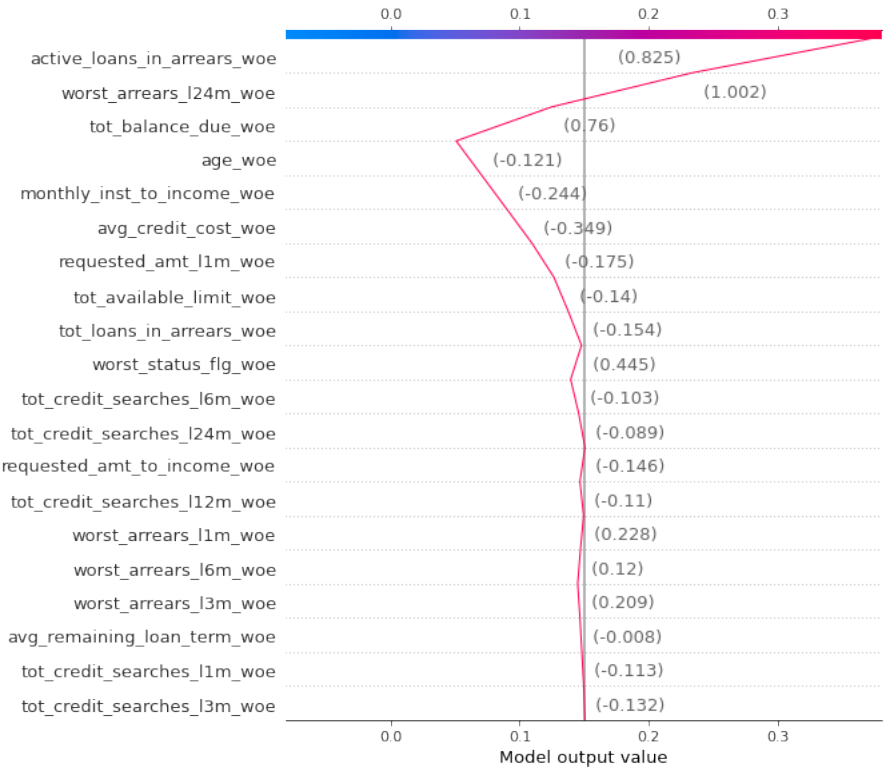


Figure 5.18: Single prediction Deep SHAP interpretation for WOE Neural Network.

5.10.6 Attributions between normal and bias class weights

Comparing the feature attributions between Figure 5.13 and 5.14, there are a few key differences. The first is that the order of the features are different which means the features which contribute the most differ. Looking at *monthly_inst_to_income* and *total_loan_in_arrears* in Figure 5.14, it can be seen that they have stronger attributions towards the *default* class in the weighted model. Another notable difference is the *worst_arrears_l24m_feature*, in Figure 5.14 the maximum value possible SHAP value is larger which means that this feature can have a larger effect on the models outcome. From this we can see that it seems that by using the misclassification costs introduced in Section 5.6.4 some of the features were given more predictive power towards the *default* class.

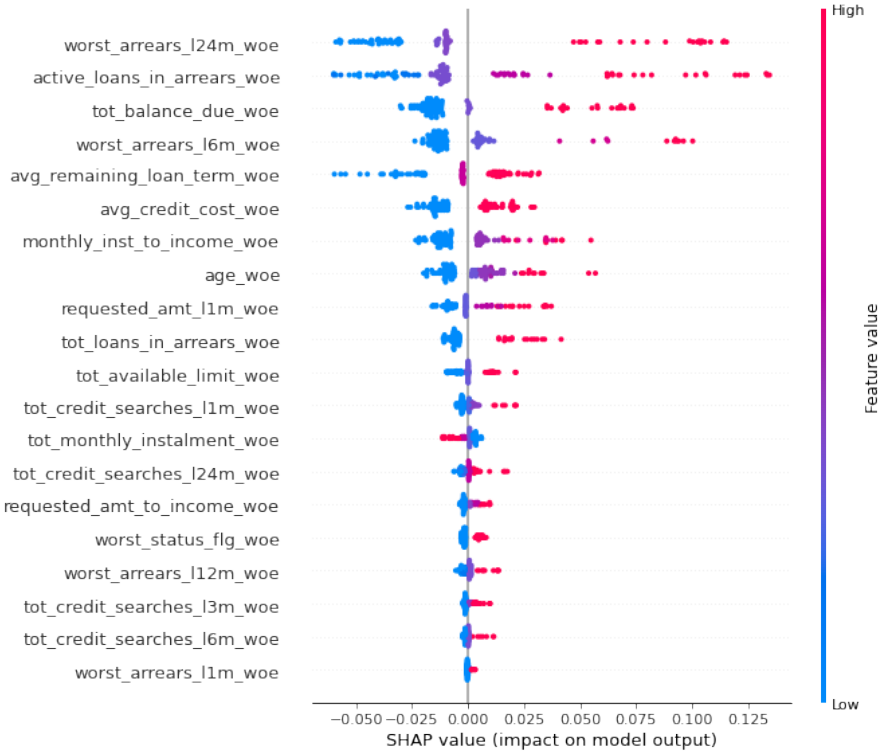


Figure 5.19: Neural Network WOE with just the input layer and output sigmoid layer.

5.10.7 Prediction strength relative to Feature magnitude

With SHAP we are able to sample the explanations over multiple instances. If the chosen background instances are diverse, we are able to view how the feature attributions react over varying magnitudes. For example if we look at the odds ratio for *worst_status_flg* in Table 5.5 we can see that the feature has an *odds ratio* of 1.069937 which by using our domain knowledge we know that this is a binary categorical variable which can only take on a value of 0 or 1. However if we did not have domain knowledge about this variable it could possibly be mistaken as a *continuous variable* which makes it seem as though it could have a much larger contribution than it actually does. Now if we compare to this to the SHAP explanation in Figure 5.13 we can see how the features contributions reacts at different values. From this we can see that regardless of it's magnitude the feature relatively has the same contribution if it is high and a small contribution when it

is low. It would require more effort and observing the original feature to discern this from just the Logistic Regression explanation. This could possibly be used for *Feature Selection*. By observing how the contribution of a feature changes over varying magnitudes we can choose to manually remove features which regardless of their intensity seem to add little value. If we once again look at Figure 5.13, the feature *tot_monthly_instalment* seems have a very low SHAP Value even when it's feature value is high and could possibly be considered for removal.

Chapter 6

Conclusion

The aim for this thesis was to provide explanations into the decisions made by Neural Networks so that they may be considered as possible solutions to problems where interpretability is a strict requirement. We discuss the findings which we have gathered during this experiment and note some work that still has to be done.

6.1 Conclusions

6.1.1 Explaining simple Neural Networks

We were able to produce explanations for various different model architectures and have also proved that the explanations generated by LIME and SHAP are nearly identical to the weight coefficients extracted from linear models. We were able to deduce how much each feature contributes relative to their magnitude and to one another. Therefore we can conclude that these tools do provide interpretations into black-box networks, however we are only able to gather a simple relationship between the input and output layers, with *Lucid* being the only tool which attempts to explain the relationship between hidden layers.

6.1.2 Lucid is complicated to use

From our evaluation of *Lucid* we have shown that it requires considerable effort and probing into the network in order to gather any form of visualization. It is

still an actively developed project and therefore there is not much documentation on how to adapt it for use with different model architectures. Although Lucid is hard to use for models which have already being built, it can be valuable while developing an image-based model to understand exactly what is happening within each layer.

6.1.3 Exposing problems within the model

As we have seen in Section 5.10.4 by using SHAP we were able to expose a misbehaving feature in our Neural Network. Although by simply viewing performance metrics, a model may seem to be performing well, by looking at the produced explanations we were able to notice that there was an issue either within our models architecture or the data itself. This is useful when testing whether a model is trustworthy enough to be used in production.

6.1.4 Comparing network architectures

As seen in Section 5.10.5 we were able to demonstrate how a change in a Neural Networks architecture affected the features which contribute towards the prediction. It is common when training a Neural Network to experiment with different hidden layers and activation functions. SHAP explanations can help us identify which variations are best suited for solving our problem.

6.1.5 Using SHAP for feature selection

Looking at Section 5.10.7 we have seen that by observing our SHAP explanations we were able to discover features which contribute little towards the prediction. Discovering variables with weak predictive strength was as simple as looking at how the SHAP value changed based on the features magnitude. This may be considered a manual process however it can scale well into many features, as it is ordered according to predictive strength. This can also be used when heavily correlated variables are discovered and a decision has to be made as to which to keep and which to remove.

6.1.6 SHAP is in most ways superior to LIME

Comparing the explanations of LIME to SHAP we can see that SHAP is more descriptive and provides more ways to express the explanations. The largest limitation of LIME is that it is only able to provide explanations for a single prediction at a time and the explanations provided can be seen as a local explanation. SHAP allows us to use a set of background samples where the SHAP values are calculated using all of these samples. SHAP also allows us to view explanations of multiple instances at a time in order to compare their magnitudes. This is to be expected since SHAP further expands upon the ideas introduced by LIME, therefore we are able to conclude that SHAP is the superior model agnostic explainer.

6.2 Future Work

6.2.1 Complex networks

All our experiments were based on simple architectures and therefore we do not know how well these tools scale for larger and more complex networks. Further experimentation has to be done in order to conclude that these tools are stable enough to be used for production models.

6.2.2 Streamlining Lucid

Lucid is the only tool which gains insight into the relationship of hidden layers. At the time of writing Lucid only works with Tensorflow 1 and is an extensive toolkit without much documentation on how to adapt it for use in custom models. It would be interesting to contribute to it's active research by providing functionality for platform independent Neural Networks and streamlining the process of interpretation.

6.2.3 Explaining hidden layers

Both LIME and SHAP are unable to provide insight into hidden layers which are arguably the most interesting part of Neural Networks. Further work could include

adapting the concepts introduced in SHAP to be able to provide explanations between any two layers within a network.

6.2.4 Quantitatively comparing different explanations

Currently we have no known way to quantitatively measure the differences between varying explanations due to different methods producing values which have different meanings. In this thesis we have resorted to using qualitative comparisons by using descriptions to explain what the different explanations mean. Further research would benefit from better methodologies being used to compare different forms of explanations.

6.2.5 Fooling LIME and SHAP

The paper *Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods* [64] provides a reason as to why the explanations of *LIME* and *SHAP* are not always to be trusted since they are prone to attacks by adversaries which aim to alter the produced explanations. A novel *scaffolding* technique is introduced which effectively hides the bias of any given classifier. Using a real world dataset known as *COMPAS* which has a inherent racist bias, the authors of the paper were able to fool LIME and SHAP into producing innocuous explanations that do not expose this bias. This is an exploit that has to be further explored and fixed before we can consider using them for models in production where they are open to attacks by adversaries.

List of References

[1] Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M. and Elhadad, N.: Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, p. 1721–1730. Association for Computing Machinery, New York, NY, USA, 2015. ISBN 9781450336642.
Available at: https://doi.org/10.1145/2783258.2788613

[2] Lipovetsky, S. and Conklin, M.: Analysis of Regression in Game Theory Approach. *Applied Stochastic Models in Business and Industry*, vol. 17, pp. 319–330, 2001.

[3] Rumelhart, D.E., Hinton, G.E. and Williams, R.J.: Learning Representations by Back-Propagating Errors. In: *Neurocomputing: Foundations of Research*, p. 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262010976.

[4] Bishop, C.M.: *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA, 1995. ISBN 0198538642.

[5] Gradient Descent and Stochastic Gradient Descent. https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/.
Available at: https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

[6] Quiza, R. and Davim, J.: Computational Methods and Optimization. In: *Machining of Hard Materials*, pp. 177–208. 2011. ISBN 978-1-84996-449-4.

[7] Montavon, G., Samek, W. and Müller, K.-R.: Methods for Interpreting and Understanding Deep Neural Networks. *CoRR*, vol. abs/1706.0, 2017.
Available at: http://arxiv.org/abs/1706.07979

[8] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R.: Intriguing properties of neural networks. 2014.

[9] Goodfellow, I.J., Shlens, J. and Szegedy, C.: Explaining and Harnessing Adversarial Examples. 2015.

[10] Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L.: Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, vol. 110, no. 3, p. 346–359, 6 2008. ISSN 1077-3142.
Available at: https://doi.org/10.1016/j.cviu.2007.09.014

[11] Wu, J., Cui, Z., Sheng, V., Zhao, P., Su, D. and Gong, S.: A Comparative Study of SIFT and its Variants. 2013.

[12] Schmidhuber, J.: Deep Learning in Neural Networks: An Overview. *CoRR*, vol. abs/1404.7, 2014.
Available at: http://arxiv.org/abs/1404.7828

[13] Hosseini, H. and Poovendran, R.: Deep Neural Networks Do Not Recognize Negative Images. *CoRR*, vol. abs/1703.0, 2017.
Available at: http://arxiv.org/abs/1703.06857

[14] Papernot, N., McDaniel, P.D., Swami, A. and Harang, R.E.: Crafting Adversarial Input Sequences for Recurrent Neural Networks. *CoRR*, vol. abs/1604.0, 2016.
Available at: http://arxiv.org/abs/1604.08275

[15] Sundararajan, M., Taly, A. and Yan, Q.: Axiomatic Attribution for Deep Networks. *CoRR*, vol. abs/1703.0, 2017.
Available at: http://arxiv.org/abs/1703.01365

[16] Cutler, A., Cutler, D. and Stevens, J.: Random Forests. In: *Machine Learning - ML*, vol. 45, pp. 157–176. 2011.

[17] Louppe, G.: Understanding Random Forests: From Theory to Practice. 2015.

[18] Random forest interpretation with scikit-learn — Diving into data. Available at: https://blog.datadive.net/random-forest-interpretation-with-scikit-learn/

[19] Ribeiro, M.T., Singh, S. and Guestrin, C.: "Why Should {I} Trust You?": Explaining the Predictions of Any Classifier. In: *Proceedings of the 22nd {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144. 2016.

[20] Lundberg, S.M. and Lee, S.-I.: A Unified Approach to Interpreting Model Predictions. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc., 2017. Available at: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[21] https://github.com/tensorflow/lucid. . Available at: https://github.com/tensorflow/lucid

[22] Simonyan, K., Vedaldi, A. and Zisserman, A.: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. 2014.

[23] Berkes, P. and Wiskott, L.: On the Analysis and Interpretation of Inhomogeneous Quadratic Forms as Receptive Fields. *Neural Comput.*, vol. 18, no. 8, p. 1868–1895, 8 2006. ISSN 0899-7667. Available at: https://doi.org/10.1162/neco.2006.18.8.1868

[24] Loh, W.-Y.: Classification and Regression Trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, pp. 14–23, 2011.

[25] Ruder, S.: An overview of gradient descent optimization algorithms. *CoRR*, vol. abs/1609.0, 2016. Available at: http://arxiv.org/abs/1609.04747

[26] Zhou, X. and Lin, H.: Sensitivity Analysis. In: Shekhar, S. and Xiong, H. (eds.), *Encyclopedia of GIS*, pp. 1046–1048. Springer US, Boston, MA, 2008. ISBN 978-0-387-35973-1.
Available at: https://doi.org/10.1007/978-0-387-35973-1_1191

[27] Craven, M.W. and Shavlik, J.: Extracting Thee-Structured Representations of Thained Networks. 1995.

[28] Lakkaraju, H., Kamar, E., Caruana, R. and Leskovec, J.: Interpretable & Explorable Approximations of Black Box Models. *CoRR*, vol. abs/1707.0, 2017.
Available at: http://arxiv.org/abs/1707.01154

[29] Lakkaraju, H., Bach, S. and Leskovec, J.: Interpretable Decision Sets: A Joint Framework for Description and Prediction. In: *KDD : proceedings. International Conference on Knowledge Discovery & Data Mining*, vol. 2016, pp. 1675–1684. 2016.

[30] Pearl, J.: *Causality: Models, Reasoning and Inference*. 2nd edn. Cambridge University Press, USA, 2009. ISBN 052189560X.

[31] Pearl, J.: The Do-Calculus Revisited. *CoRR*, vol. abs/1210.4, 2012.
Available at: http://arxiv.org/abs/1210.4852

[32] Chattopadhyay, A., Manupriya, P., Sarkar, A. and Balasubramanian, V.N.: Neural Network Attributions: {A} Causal Perspective. *CoRR*, vol. abs/1902.0, 2019.
Available at: http://arxiv.org/abs/1902.02302

[33] Rubin, D.B.: Bayesian Inference for Causal Effects: The Role of Randomization. *Ann. Statist.*, vol. 6, no. 1, pp. 34–58, 1978.
Available at: https://doi.org/10.1214/aos/1176344064

[34] Yosinski, J., Clune, J., Nguyen, A.M., Fuchs, T.J. and Lipson, H.: Understanding Neural Networks Through Deep Visualization. *CoRR*, vol. abs/1506.0, 2015.
Available at: http://arxiv.org/abs/1506.06579

[35] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[36] Shrikumar, A., Greenside, P., Shcherbina, A. and Kundaje, A.: Not Just a Black Box: Learning Important Features Through Propagating Activation Differences. *CoRR*, vol. abs/1605.0, 2016.
Available at: http://arxiv.org/abs/1605.01713

[37] Shrikumar, A., Greenside, P. and Kundaje, A.: Learning Important Features Through Propagating Activation Differences. *CoRR*, vol. abs/1704.0, 2017.
Available at: http://arxiv.org/abs/1704.02685

[38] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R. and Samek, W.: On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PloS one*, vol. 10, no. 7, pp. e0130140–e0130140, 7 2015. ISSN 1932-6203.
Available at: https://pubmed.ncbi.nlm.nih.gov/26161953https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4498753/

[39] Montavon, G., Bach, S., Binder, A., Samek, W. and Müller, K.-R.: Explaining NonLinear Classification Decisions with Deep Taylor Decomposition. *CoRR*, vol. abs/1512.0, 2015.
Available at: http://arxiv.org/abs/1512.02479

[40] Ribeiro, M.T., Singh, S. and Guestrin, C.: Nothing else matters: model-agnostic explanations by identifying prediction invariance. *arXiv preprint arXiv:1611.05817*, 2016.

[41] Ren, X. and Malik, J.: Learning a Classification Model for Segmentation. In: *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, p. 10. IEEE Computer Society, USA, 2003. ISBN 0769519504.

[42] Vedaldi, A. and Soatto, S.: Quick Shift and Kernel Methods for Mode Seeking. In: Forsyth, D., Torr, P. and Zisserman, A. (eds.), *Computer Vision – ECCV*

2008, pp. 705–718. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-88693-8.

[43] https://github.com/marcotcr/lime. .
Available at: https://github.com/marcotcr/lime

[44] Feige, U.: A Threshold of Ln ¡i¿n¡/i¿ for Approximating Set Cover. *J. ACM*, vol. 45, no. 4, p. 634–652, 7 1998. ISSN 0004-5411.
Available at: https://doi.org/10.1145/285055.285059

[45] Kubat, M.: Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. *Fourteenth International Conference on Machine Learning*, 2000.

[46] Datta, A., Sen, S. and Zick, Y.: Algorithmic Transparency via Quantitative Input Influence: Theory and Experiments with Learning Systems. pp. 598–617. 2016.

[47] Shapley, L.S.: A value for n-person games. Tech. Rep., Rand Corp Santa Monica CA, 1952.

[48] Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K. and Mordvintsev, A.: The Building Blocks of Interpretability. *Distill*, 2018.

[49] Kadir, T. and Brady, M.: Saliency, Scale and Image Description. *Int. J. Comput. Vision*, vol. 45, no. 2, p. 83–105, 11 2001. ISSN 0920-5691.
Available at: https://doi.org/10.1023/A:1012460413855

[50] http://lib.stat.cmu.edu/datasets/boston. .
Available at: http://lib.stat.cmu.edu/datasets/boston

[51] Galea, A. and Capelo, L.: *Applied Deep Learning with Python: Use Scikit-Learn, TensorFlow, and Keras to Create Intelligent Systems and Machine Learning Solutions*. Packt Publishing, 2018. ISBN 9781789804744.

[52] Wang, W. and Lu, Y.: Analysis of the Mean Absolute Error ({MAE}) and the Root Mean Square Error ({RMSE}) in Assessing Rounding Model. {*IOP*} *Conference Series: Materials Science and Engineering*, vol. 324, p. 12049, 3

2018.
Available at: https://doi.org/10.1088%2F1757-899x%2F324%2F1%2F012049

[53] Harrison, D. and Rubinfeld, D.L.: Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, vol. 5, no. 1, pp. 81–102, 1978. ISSN 0095-0696.
Available at: http://www.sciencedirect.com/science/article/pii/0095069678900062

[54] Hassibi, K.: Detecting Payment Card Fraud with Neural Networks. In: *Business Applications of Neural Networks*, pp. 141–157.
Available at: https://www.worldscientific.com/doi/abs/10.1142/9789812813312_0009

[55] Siddiqi, N.: Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. 2005.

[56] Kullback, S. and Leibler, R.A.: On information and sufficiency. *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[57] Hastie, T., Tibshirani, R. and Friedman, J.H.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN 9780387848846.
Available at: https://books.google.co.za/books?id=eBSgoAEACAAJ

[58] Štrumbelj, E. and Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, vol. 41, pp. 647–665, 2013.

[59] Brause, R., Langsdorf, T. and Hepp, M.: Neural data mining for credit card fraud detection. In: *Proceedings 11th International Conference on Tools with Artificial Intelligence*, pp. 103–106. 11 1999. ISSN 1082-3409.

[60] Vinciotti, V. and Hand, D.: Scorecard construction with unbalanced class sizes. *Journal of the Iranian Statistical Society*, vol. 2, pp. 189–205, 2003.

[61] Russell, S. and Norvig, P.: *Artificial Intelligence: A Modern Approach*. 3rd edn. Prentice Hall Press, USA, 2009. ISBN 0136042597.

[62] Molnar, C.: *Interpretable Machine Learning*. 2019.

[63] Szumilas, M.: Explaining odds ratios. *Journal of the Canadian Academy of Child and Adolescent Psychiatry = Journal de l'Academie canadienne de psychiatrie de l'enfant et de l'adolescent*, vol. 19, no. 3, pp. 227–229, 8 2010. ISSN 2293-6122.
Available at: https://pubmed.ncbi.nlm.nih.gov/20842279https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2938757/

[64] Slack, D., Hilgard, S., Jia, E., Singh, S. and Lakkaraju, H.: Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods. 2020.